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] )          tuple
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] )
```

Singleton element

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

single [‘4’] ➡ [_]

single [‘1’..’9’] =
[‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’] ➡ _

[_, _, _, _, _, _, _, _, _] first no match

_____ second match

Single cell expansion

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

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

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

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

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

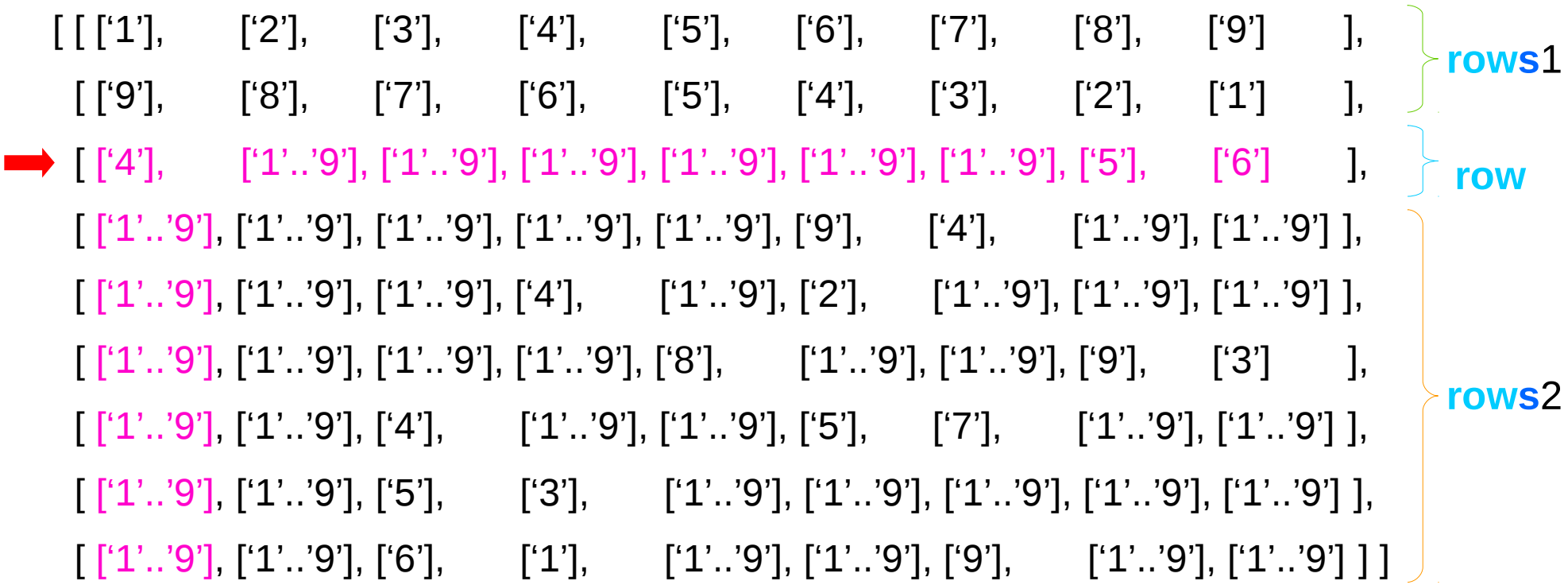
tt returns the same matrix

```
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

map (not . single)

(**rows**1, **row**:**rows**2)
(**row**1, **cs**:**row**2)



(rows1, row:rows2)

(rows1, row:rows2) = **break** (**any** (**not** . single)) **rows**

```
[ [ [‘1’],      [‘2’],      [‘3’],      [‘4’],      [‘5’],      [‘6’],      [‘7’],      [‘8’],      [‘9’],      ],
  [ [‘9’],      [‘8’],      [‘7’],      [‘6’],      [‘5’],      [‘4’],      [‘3’],      [‘2’],      [‘1’],      ] ]      rows1
[ [ [‘1’..’9’], [‘1’..’9’], [‘4’],      [‘1’..’9’], [‘1’..’9’], [‘5’],      [‘7’],      [‘1’..’9’], [‘1’..’9’] ] ]      row
[ [ [‘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’] ],      rows2
  [ [‘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’] ] ]
```

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

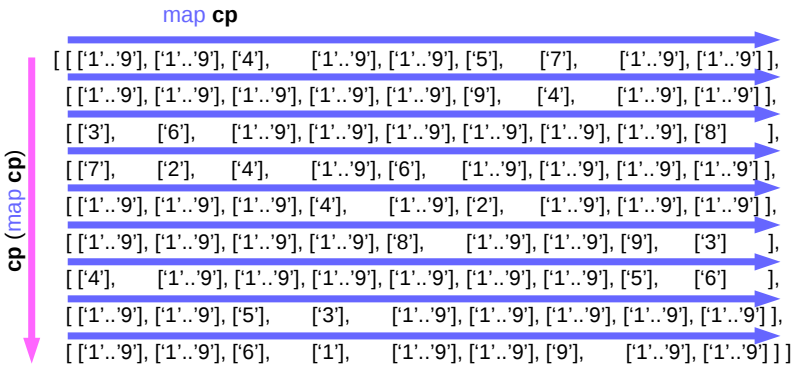
(row1, cs:row2) = **break** (not . single) **row**

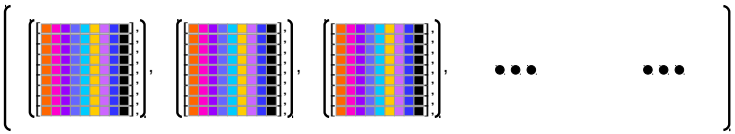
[[['1'], ['2'], ['3'], ['4'], ['5'], ['6'], ['7'], ['8'], ['9'],],
[['9'], ['8'], ['7'], ['6'], ['5'], ['4'], ['3'], ['2'], ['1'],],
rows1
[['4'], ['1'..'9'], ['1'..'9'], ['1'..'9'], ['1'..'9'], ['1'..'9'], ['1'..'9'], ['5'], ['6']],
[['1'..'9'], ['1'..'9'], ['1'..'9'], ['1'..'9'], ['1'..'9'], ['9'], ['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']],
rows2
[['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']]]

[['4'],], ['1'..'9'], [['1'..'9'], ['1'..'9'], ['1'..'9'], ['1'..'9'], ['1'..'9'], ['5'], ['6']]

row1 **cs** **row**2

expand





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

Matrix Choices

Matrix [Digit]

expand

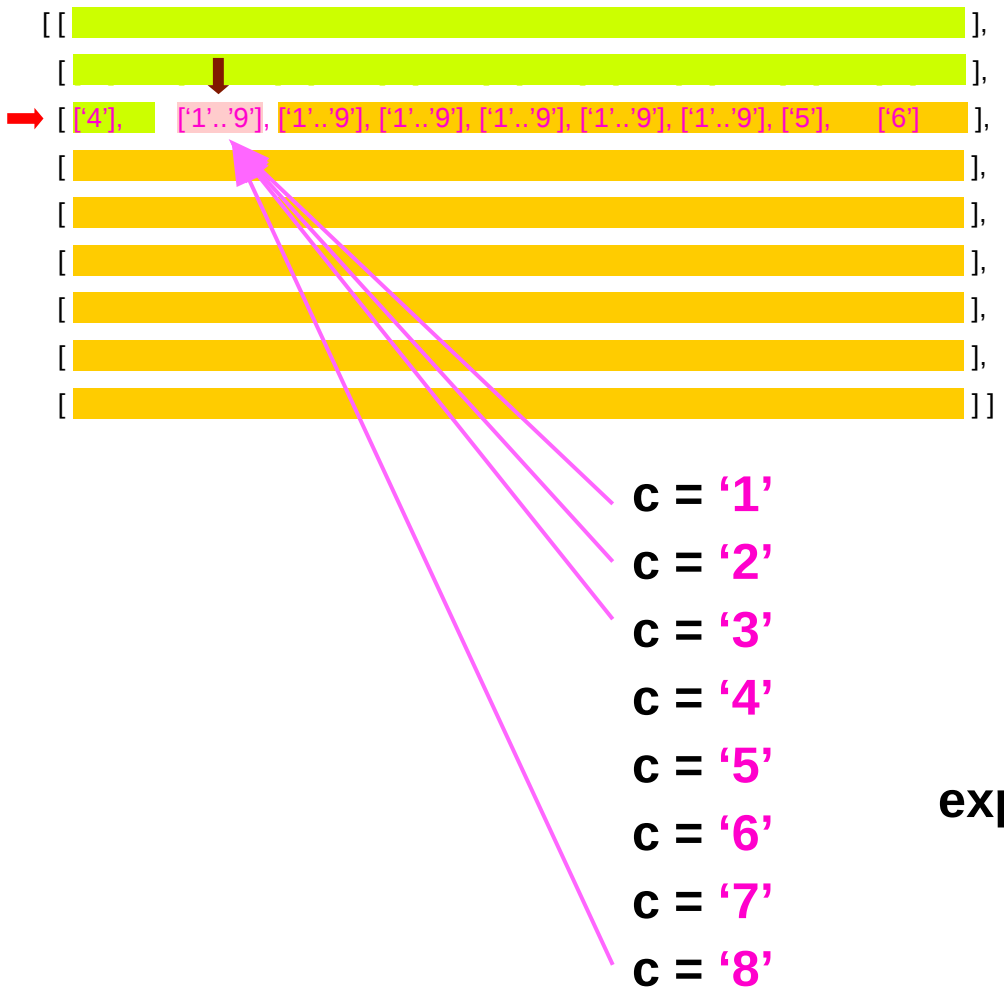
[Grid]

[Matrix Digit]

expand = concat . map expand . expand1

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

expand1



Partial Expansion + Pruning

Single Cell Expansion + Pruning

→ hope to improve the speed

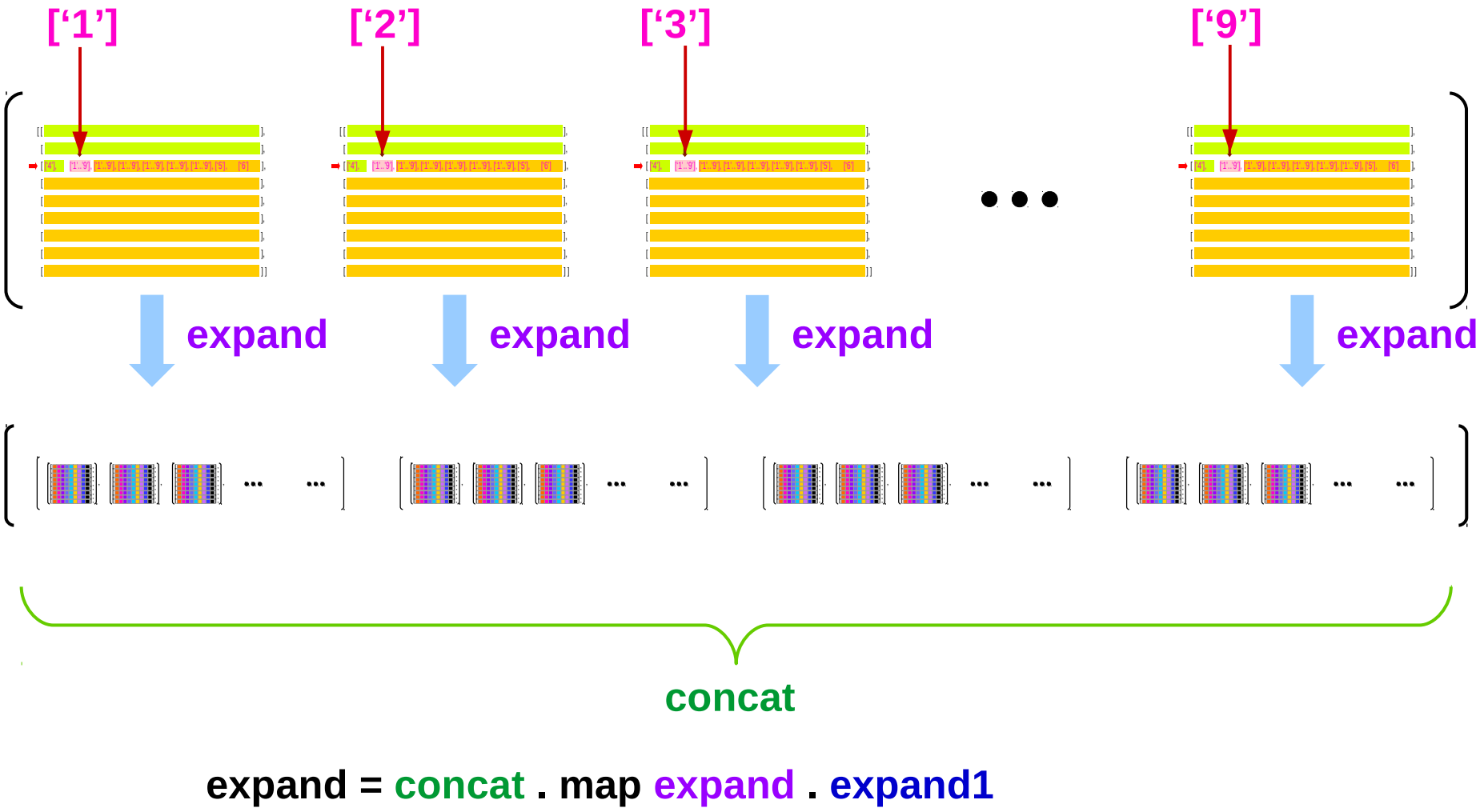
expand1

- First find a non-singleton element
- Perform single cell expansion
  over the found non-singleton
- Then do the regular **expand**
- Then **concat** the results (combine)

expand = concat . map expand . expand1

expand1

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




```
expand = concat . map expand . expand1
```

expand1 - scheme 1

cs = ['1'..'9']

any non-singleton element example

```
[ [rows1 ++ [row1 ++ ['1']:row2] ++ rows2] ,      c = '1'
  [rows1 ++ [row1 ++ ['2']:row2] ++ rows2] ,      c = '2'
  [rows1 ++ [row1 ++ ['3']:row2] ++ rows2] ,      c = '3'
  [rows1 ++ [row1 ++ ['4']:row2] ++ rows2] ,      c = '4'
  [rows1 ++ [row1 ++ ['5']:row2] ++ rows2] ,      c = '5'
  [rows1 ++ [row1 ++ ['6']:row2] ++ rows2] ,      c = '6'
  [rows1 ++ [row1 ++ ['7']:row2] ++ rows2] ,      c = '7'
  [rows1 ++ [row1 ++ ['8']:row2] ++ rows2] ]      c = '8'
```

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

expand1 - scheme 1

```
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
```

```
cs = ['1'..'9']
```
assumed

```
[rows1 ++ [row1 ++ [c]:row2] ++ rows2]
```

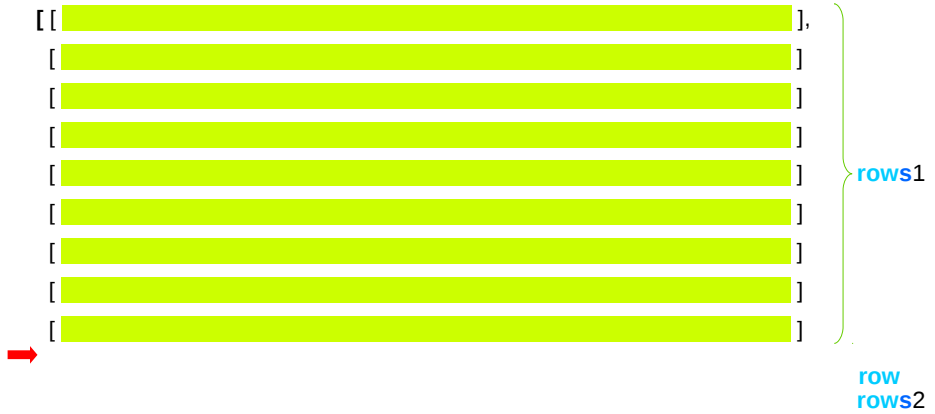
expand1 - scheme 1

```
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
```

If all the elements of a matrix are singletons, then
break (**any** (**not . single**)) **rows** → [**rows**, [ ]]
And **expand1** returns error

Also, this **expand1** is not efficient

(rows1, row:rows2)



If all the elements of a matrix are singletons, then

break (**any** (**not .** **single**)) **rows** → [**rows**, [ ]]

and **expand1** returns error

also, this **expand1** is not efficient

Identifying quickly the case of no solution

Non-singleton entry is an empty list

expand1 will return the empty list

expand1 will waste its work

If it is buried deep in the matrix

Identifying quickly the case of no solution

expand1

- First find an entry with the smallest number of choices (not equal to 1)
- Perform single cell expansion over the selected entry
- Then do the regular **expand**
- Then **concat** the results (combine)

This may be a better choice of cell on which to perform expansion
A cell with no choices means that the puzzle is unsolvable
It enables us to figure out such a cell quickly

expand1 - scheme 2

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

expand1 - scheme 2

concat rows

```
[ [ ['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'] ] ]
```

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

map length . concat rows

map length . concat rows

```
[ ['1'], ['1'], ['1'], ['1'], ['1'], ['1'], ['1'], ['1'], ['1']  ,
  ['1'], ['1'], ['1'], ['1'], ['1'], ['1'], ['1'], ['1'], ['1']  ,
  ['1'], ['9'..'9'], ['9'..'9'], ['9'..'9'], ['9'..'9'], ['9'..'9'], ['9'..'9'], ['1'], ['1']  ,
  ['9'..'9'], ['9'..'9'], ['9'..'9'], ['9'..'9'], ['9'..'9'], ['1'], ['1'], ['9'..'9'], ['9'..'9']  ,
  ['9'..'9'], ['9'..'9'], ['9'..'9'], ['1'], ['9'..'9'], ['1'], ['9'..'9'], ['9'..'9'], ['9'..'9']  ,
  ['9'..'9'], ['9'..'9'], ['9'..'9'], ['9'..'9'], ['1'], ['9'..'9'], ['9'..'9'], ['1'], ['1']  ,
  ['9'..'9'], ['9'..'9'], ['1'], ['9'..'9'], ['9'..'9'], ['1'], ['1'], ['9'..'9'], ['9'..'9']  ,
  ['9'..'9'], ['9'..'9'], ['1'], ['1'], ['9'..'9'], ['9'..'9'], ['9'..'9'], ['9'..'9'], ['9'..'9'] ,,
  ['9'..'9'], ['9'..'9'], ['1'], ['1'], ['9'..'9'], ['9'..'9'], ['1'], ['9'..'9'], ['9'..'9'] ]
```

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

counts rows

filter (/= 1) . map length . concat rows

```
[
        9 ,   9 ,   9 ,   9 ,   9 ,   9 ,
   9 ,   9 ,   9 ,   9 ,   9 ,           9 ,   9    ,
   9 ,   9 ,   9 ,         9 ,         9 ,   9 ,   9    ,
   9 ,   9 ,   9 ,   9 ,         9 ,   9 ,
   9 ,   9 ,         9 ,   9 ,           9 ,   9    ,
   9 ,   9 ,               9 ,   9 ,   9 ,   9 ,   9    ,
   9 ,   9 ,               9 ,   9 ,         9 ,    9    ]
```

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

Matrix Digit & Matrix Choices

```
[ [ '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' ] ]
```

```
[ [ ['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 Digit = [Row Digit] = [[Digit]]

= Grid

Matrix [Digit] = [Row [Digit]] = [[[Digit]]]

= Matrix Choices

Type Definitions

```
type    Digit       =   Char                ‘1’
type    Choices     =   [Digit]             [‘1’ , ‘2’, ‘3’]

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

        Matrix Digit      =   [Row Digit]      =   [[Digit]]
        Matrix [Digit]    =   [Row [Digit]]    =   [[[Digit]]]

        Matrix Choices    =   [Row Choices]    =   [[Choices]] = [[[Digit]]]
```

[Row Digit] and [Row [Digit]]

```
type Row a = [a]

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

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

Grid and Choices

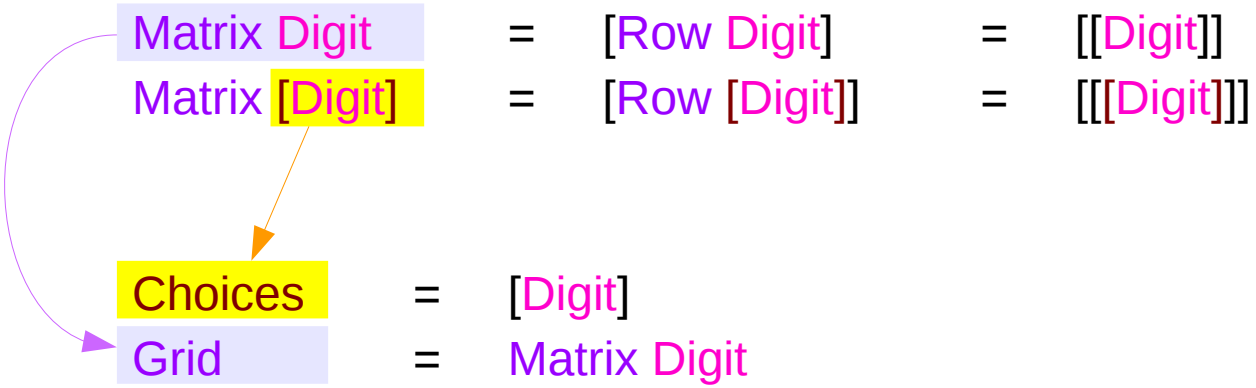
Grid

```
[ [ '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' ] ]
```

```
[ [ ['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 Digit = [Row Digit] ➡ [[Digit]]

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



complete

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

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

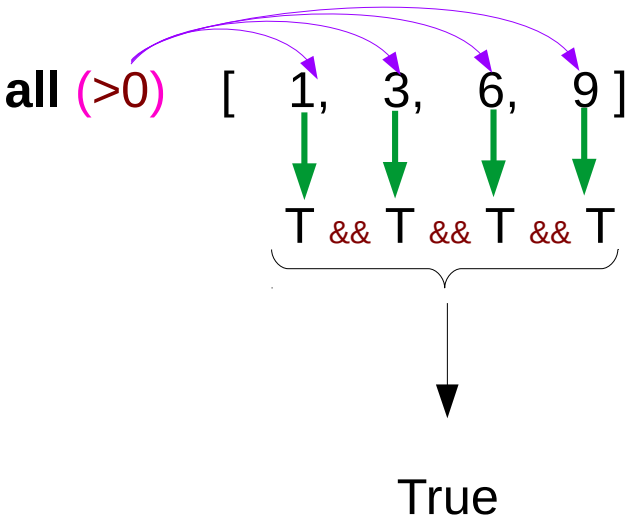
single [‘4’] → True
Single [‘1’..’9’] → False

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

all

all :: (a -> Bool) -> [a] -> Bool

Determines whether all elements of the structure satisfy the predicate.



safe, ok

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

```
ok row = nodups [x | [x] <- row]
```

```
[ [ ['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]]]

Matrix [Digit] = Matrix Choices

```
[x | x <- [0..100], odd x]
{x | x ∈ [0..100], odd x}
```

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

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

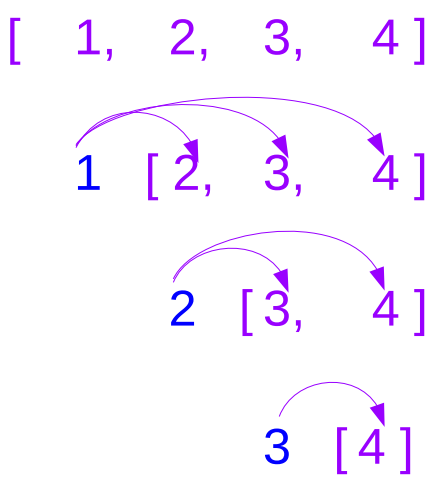
valid, nodups

```
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
```

Matrix Digit = Grid

```
[ [ '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' ] ]
```



complete, safe

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

safe :: Matrix [Digit] -> Bool
safe cm =    all ok (rows cm)&&
             all ok (cols cm) &&
             all ok (boxs cm)

ok row = nodups [x | [x] <- row]
```

complete, safe

A safe matrix: No duplicates

- Singleton choices do not have duplicates in any row, column, or box.
- But it can have non-singleton choices
- Pruning can introduce unsafe matrices.
- But if a matrix is safe after pruning, it must be safe before pruning.
- **safe** . **prune** = **safe**

A complete matrix: All singletons

- **No** non-singleton choices in any row, column, or box.
- A solution must be **safe** and **complete**

Expand Types

map **cp**

cp (map **cp**)

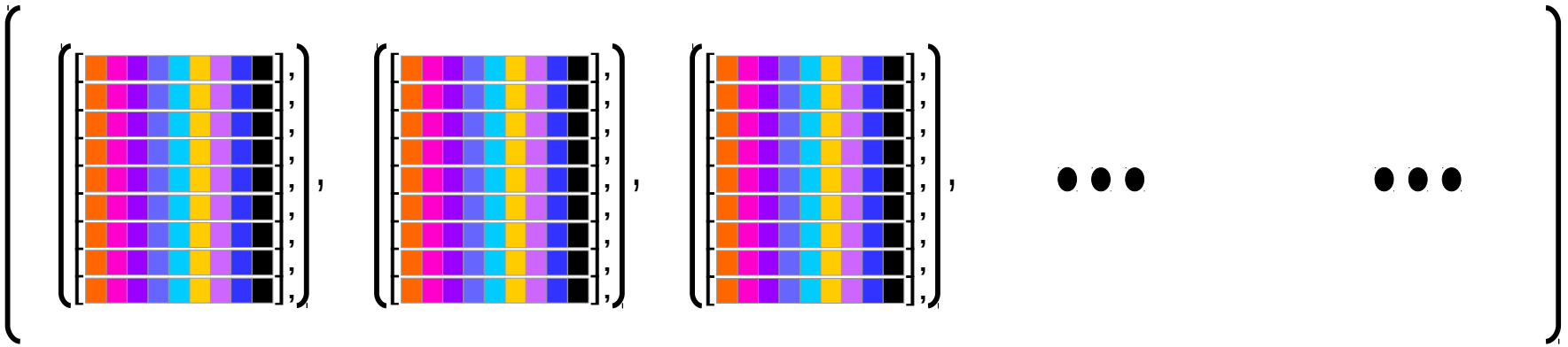
[[[‘1’..’9’], [‘1’..’9’], [‘4’], [‘1’..’9’], [‘1’..’9’], [‘5’], [‘7’], [‘1’..’9’], [‘1’..’9’]],
[[‘1’..’9’], [‘1’..’9’], [‘1’..’9’], [‘1’..’9’], [‘1’..’9’], [‘9’], [‘4’], [‘1’..’9’], [‘1’..’9’]],
[[‘3’], [‘6’], [‘1’..’9’], [‘1’..’9’], [‘1’..’9’], [‘1’..’9’], [‘1’..’9’], [‘1’..’9’], [‘8’]],
[[‘7’], [‘2’], [‘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 **[**Digit**]** = Matrix Choices

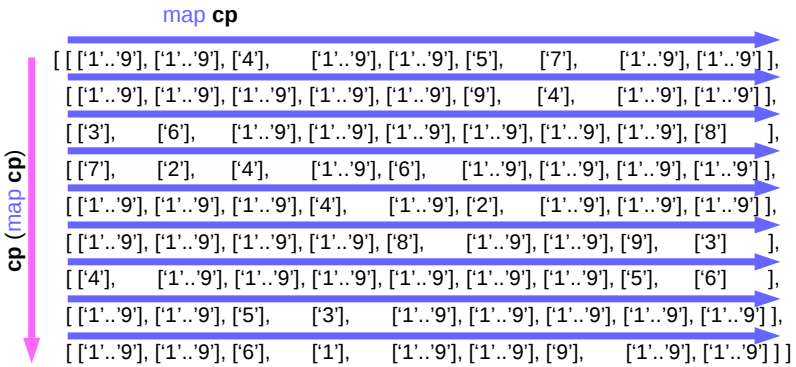
[[‘0’, ‘0’, ‘4’, ‘0’, ‘0’, ‘5’, ‘7’, ‘0’, ‘0’],
[‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘9’, ‘4’, ‘0’, ‘0’],
[‘3’, ‘6’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘8’],
[‘7’, ‘2’, ‘4’, ‘0’, ‘6’, ‘0’, ‘0’, ‘0’, ‘0’],
[‘0’, ‘0’, ‘0’, ‘4’, ‘0’, ‘2’, ‘0’, ‘0’, ‘0’],
[‘0’, ‘0’, ‘0’, ‘0’, ‘8’, ‘0’, ‘0’, ‘9’, ‘3’],
[‘4’, ‘0’, ‘0’, ‘0’,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

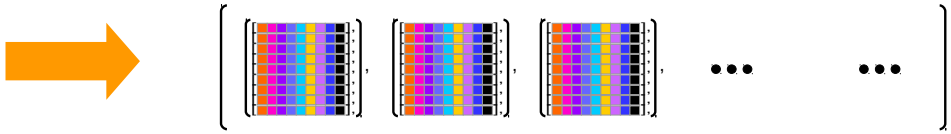
Matrix Digit = Grid

[Matrix Digit**]** = **[**Grid**]**



expand





Matrix Choices

Matrix [Digit]

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

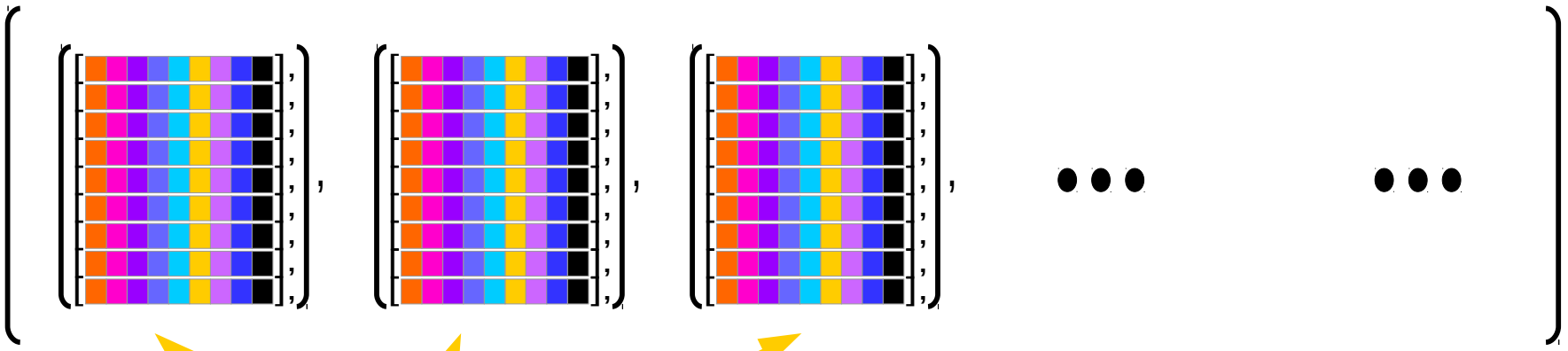
[Grid]

[Matrix Digit]

filter valid (expand m)

filter valid (**expand** m)

[Matrix Digit] = [Grid]



filter

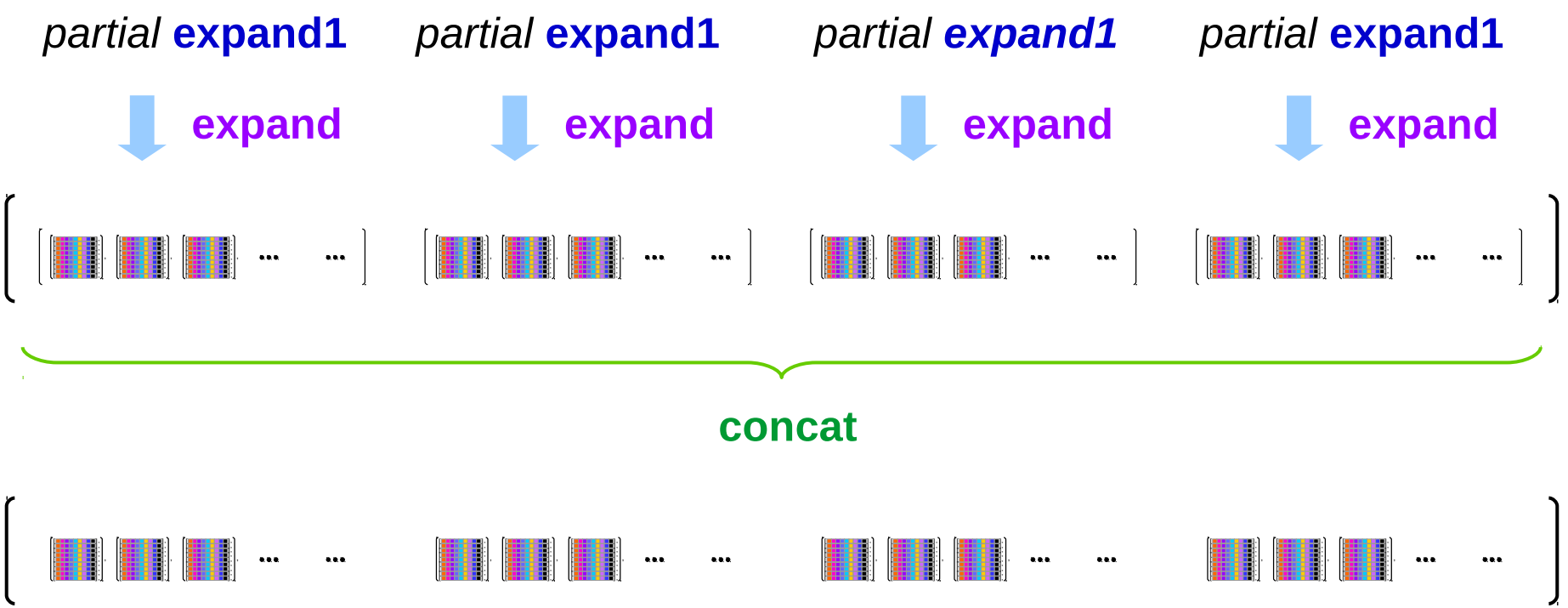
```
valid  :: Grid -> Bool
valid g = all nodups (rows g) &&
          all nodups (cols  g) &&
          all nodups (boxs g)
```

Matrix Digit = Grid

```
[ [ '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' ] ]
```

concat . map **expand** . expand1

expand = concat . map expand . expand1



filter valid (expand m)

For a **safe** and **complete** matrix:

filter valid (**expand** m)

➡ [**extract** m]

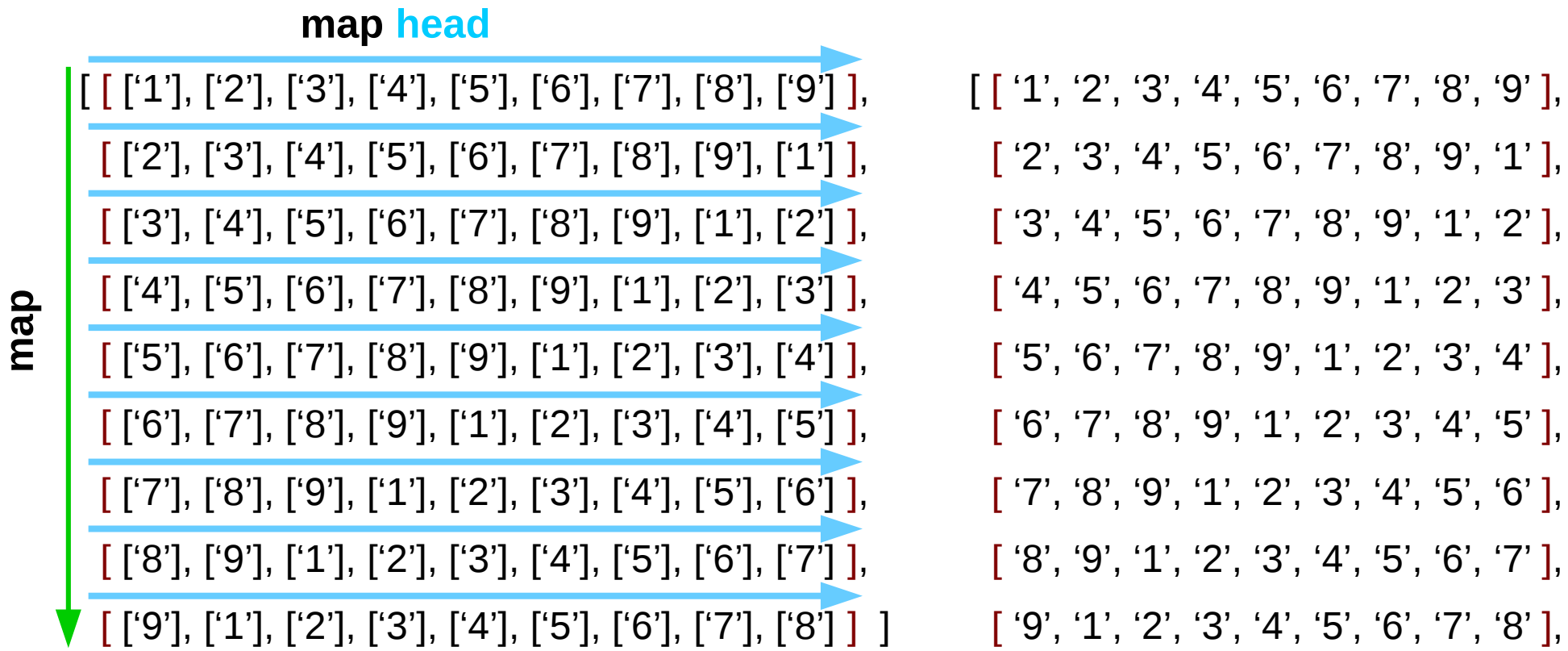
For a **safe** and **incomplete** matrix:

filter valid (**expand** m)

➡ **filter valid** . (**concat** (**map expand** (**expand1** m)))

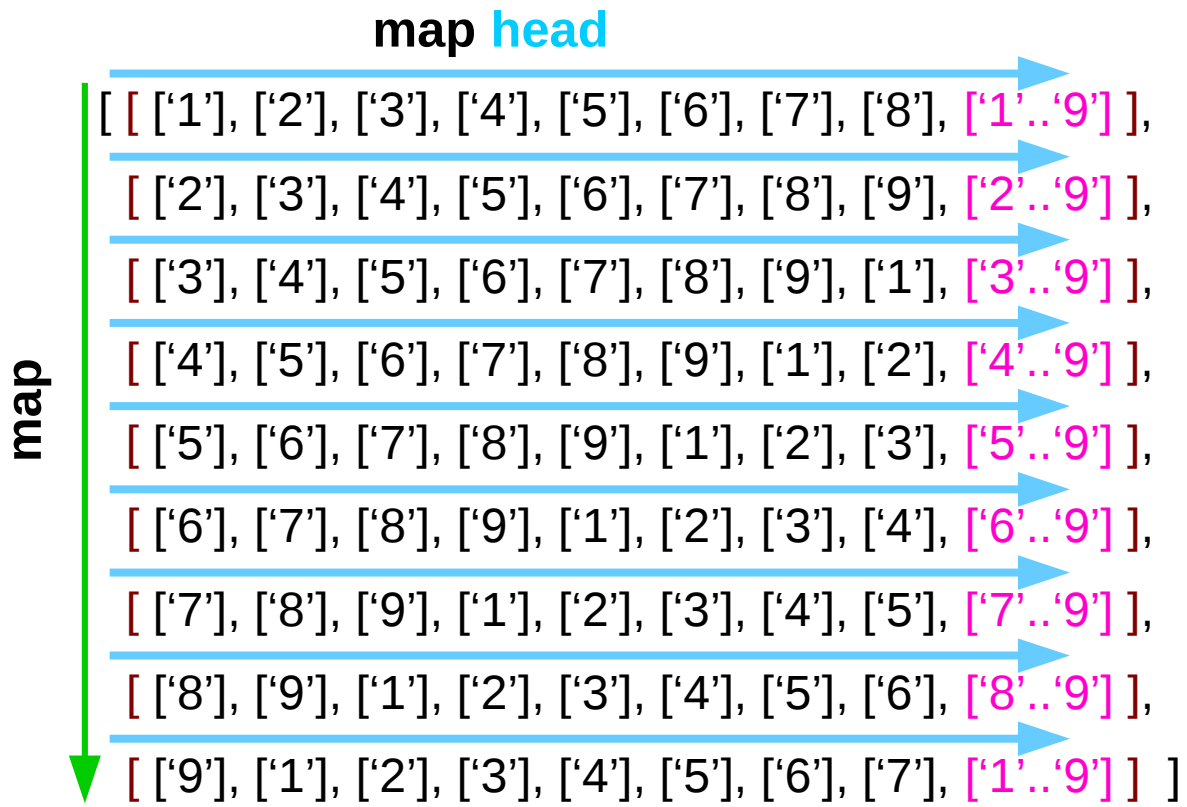
Safe and Complete Matrix Example

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



Safe and Incomplete Matrix Example

filter valid . (**concat** (**map expand** (**expand1** m)))



non-singleton choices
are allowed

Point free style

```
fn x = ceiling (negate (tan (cos (max 50 x))))

fn = ceiling . negate . tan . cos . max 50
```

filter valid . expand

filter valid . expand

expand = concat . map expand . expand1

= filter valid . concat . map expand . expand1

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

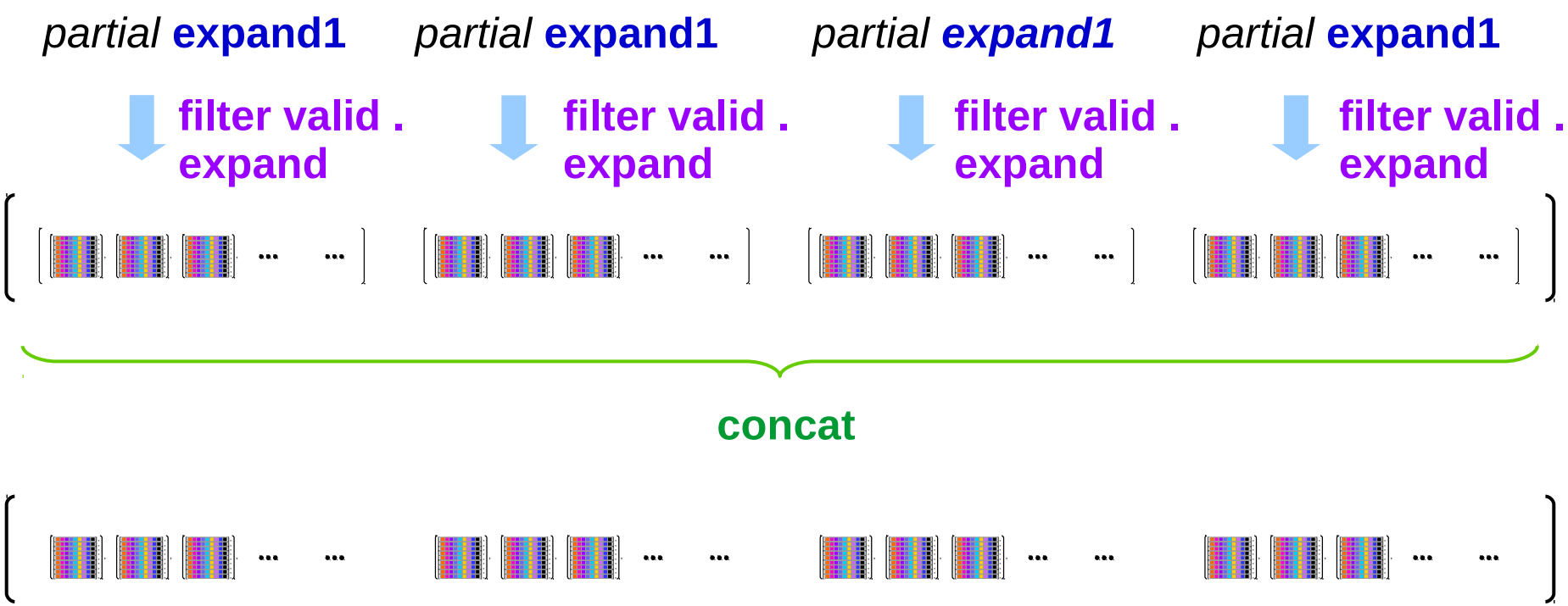
= concat . map (filter valid) . map expand . expand1

map (filter p) . map expand = map (filter p . expand)

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

concat . map (**filter valid** . **expand**) . expand1

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



Various valid expansion

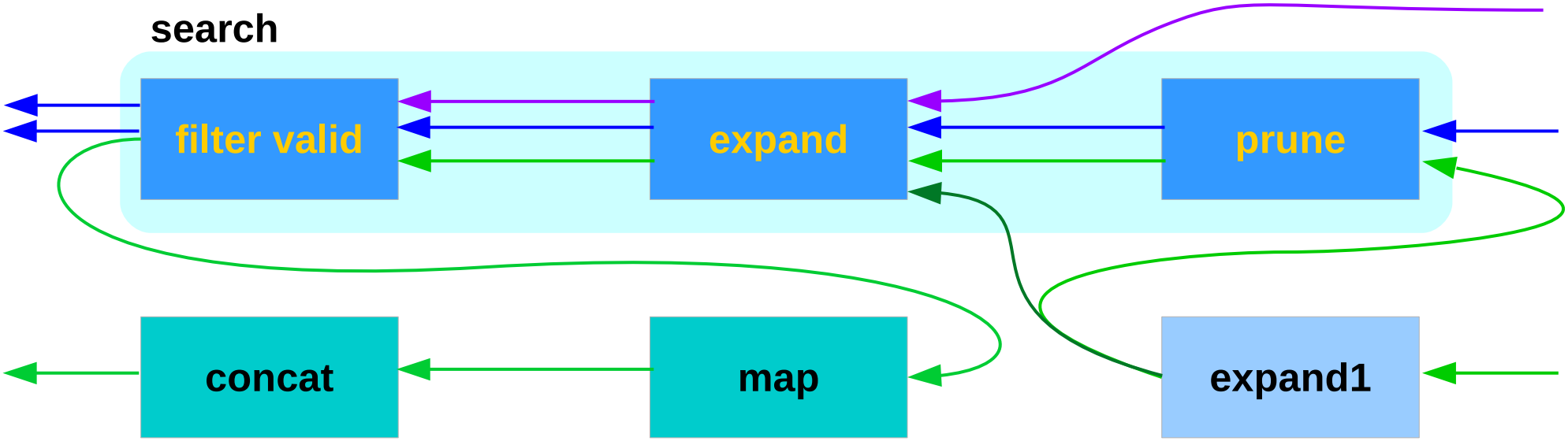
filter valid . expand
filter valid . expand . prune
search

concat . map (filter valid . expand). expand1
concat . map (filter valid . expand . prune) . expand1
concat . map search . expand1

search = filter valid . expand . prune

extract

filter valid . expand
filter valid . expand . prune
concat . map (filter valid . expand) . expand1
concat . map (filter valid . expand . prune) . expand1



Partial expansion version of expand

filter valid . expand
= concat . map (filter valid . expand) . expand1
partial expansion version

filter valid . expand . prune
= concat . map (filter valid . expand . prune) . expand1
partial expansion version

search = filter valid . expand . prune

search
= concat . map search . expand1
partial expansion version

search

search = filter valid . expand . prune

For a **safe** and **incomplete** matrix:

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

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

filter valid . expand . prune
= concat . map (filter valid . expand . prune) . expand1

expand = concat . map expand . expand1

solve

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

solve = search . choices
```

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

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

Matrix Digit & Matrix Choices

```
[ [ '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' ] ]
```

```
[ [ ['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 Digit = [Row Digit] = [[Digit]]

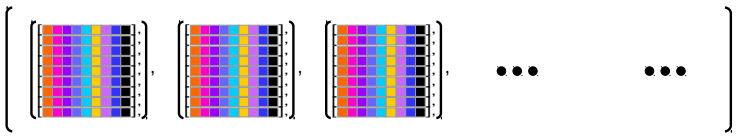
= Grid

solve2 :: Grid -> [Grid]

Matrix [Digit] = [Row [Digit]] = [[[Digit]]]

= Matrix Choices

search :: Matrix Choices -> [Grid]



many

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**

Apply prune recursively until no more pruned matrix.

```
y = many prune x
      y1 = prune x
      y2 = prune y1
      y3 = prune y2
       ...
      yn = prune yn-1
```

choices function examples

		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' ] ]
```

```
[ [ ['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'] ] ]
```

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

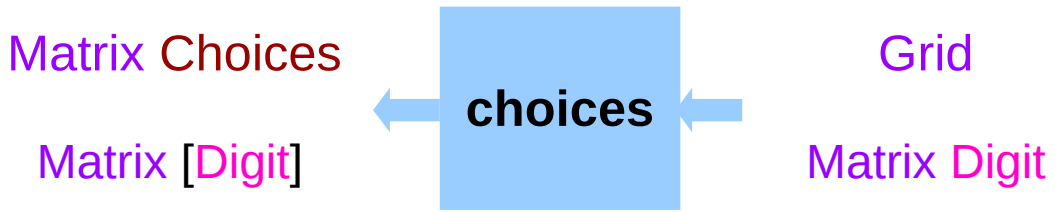
Function: choices

```
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



expand1

```
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]
```

break, any, or

```
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
```

expand1 ver 1.

```
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, []]
```

expand1 ver 2.

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

complete, safe

```
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]
```

extract

```
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
```

solve

```
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