A Sudoku Solver – Rules (2A)

- 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

Lawful laws (1)

rows . **rows** = id
cols . **cols** = id
boxs . **boxs** = id

ungroup . group = id
group . ungroup = id

map ungroup . ungroup . map **cols** . group .
ungroup . map **cols** . group . map group =
.
map ungroup . ungroup . map **cols** .
map **cols** . group . map group =

map ungroup . ungroup .
group . map group = id

Lawful laws (2)

map **rows** . **expand** = **expand** . **rows**
map **cols** . **expand** = **expand** . **cols**
map **boxs** . **expand** = **expand** . **boxes**

map (map f) . cp = cp . map (map f)
filter (all p) . cp = cp . map (filter p)

rows, cols, boxes composite functions

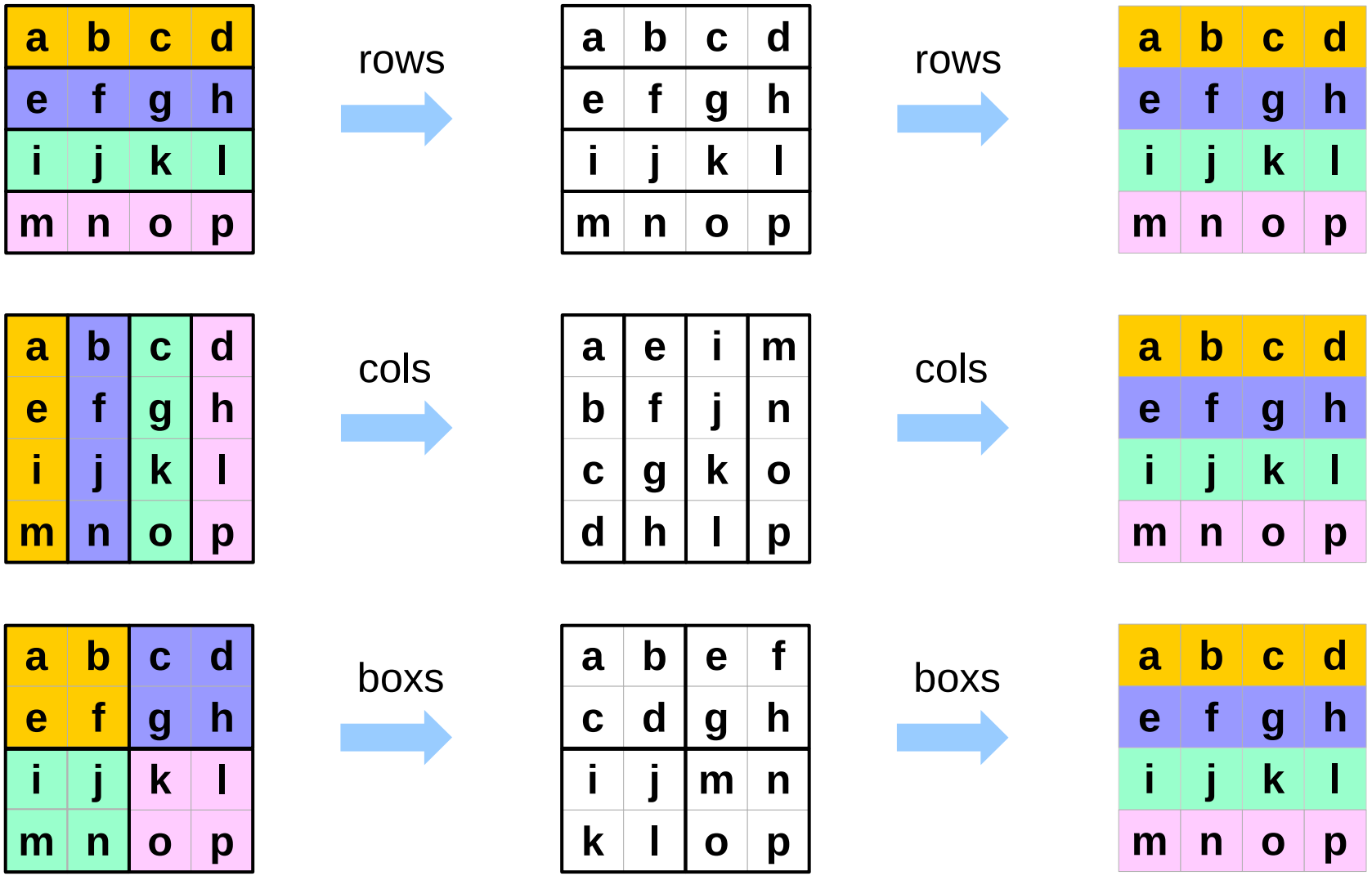
rows . **rows** = id
cols . **cols** = id
boxs . **boxs** = id

```
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
```

rows.rows, cols.cols, boxs.boxs



group and ungroup

ungroup. group = id
group . ungroup = id

ungroup = concat

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

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

group . ungroup

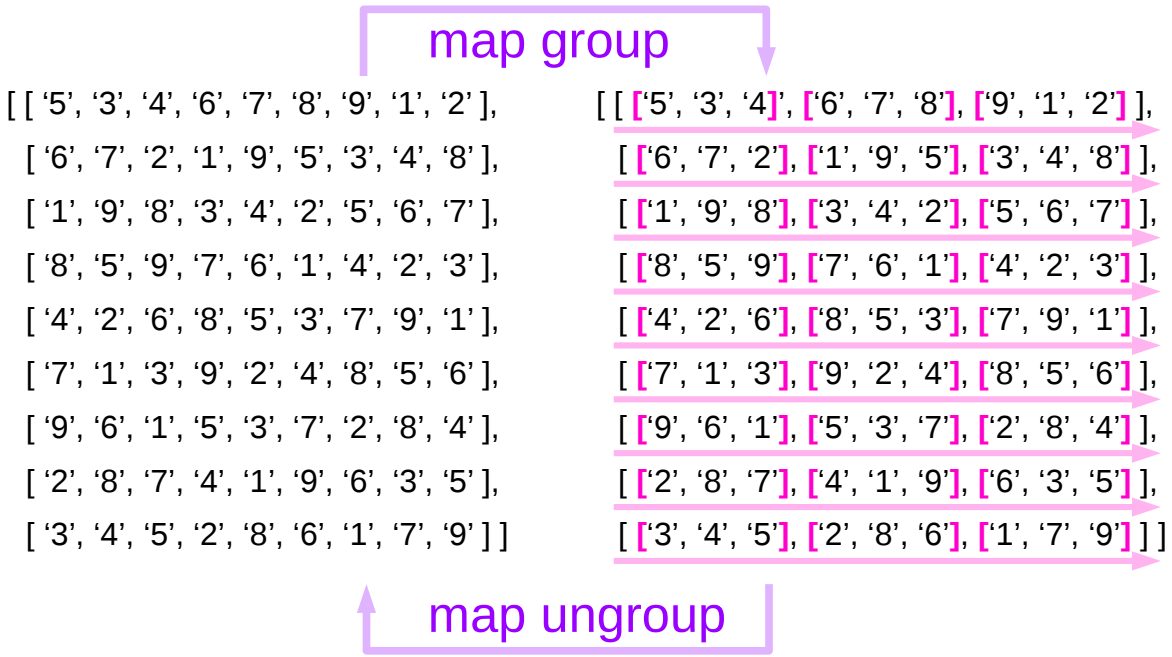
```
[ [ ‘5’, ‘3’, ‘4’, ‘6’, ‘7’, ‘8’, ‘9’, ‘1’, ‘2’ ],
  [ ‘6’, ‘7’, ‘2’, ‘1’, ‘9’, ‘5’, ‘3’, ‘4’, ‘8’ ],
  [ ‘1’, ‘9’, ‘8’, ‘3’, ‘4’, ‘2’, ‘5’, ‘6’, ‘7’ ],
  [ ‘8’, ‘5’, ‘9’, ‘7’, ‘6’, ‘1’, ‘4’, ‘2’, ‘3’ ],
  [ ‘4’, ‘2’, ‘6’, ‘8’, ‘5’, ‘3’, ‘7’, ‘9’, ‘1’ ],
  [ ‘7’, ‘1’, ‘3’, ‘9’, ‘2’, ‘4’, ‘8’, ‘5’,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

group

ungroup

```
[ [ [ ‘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’ ] ] ]
```

map ungroup . map group

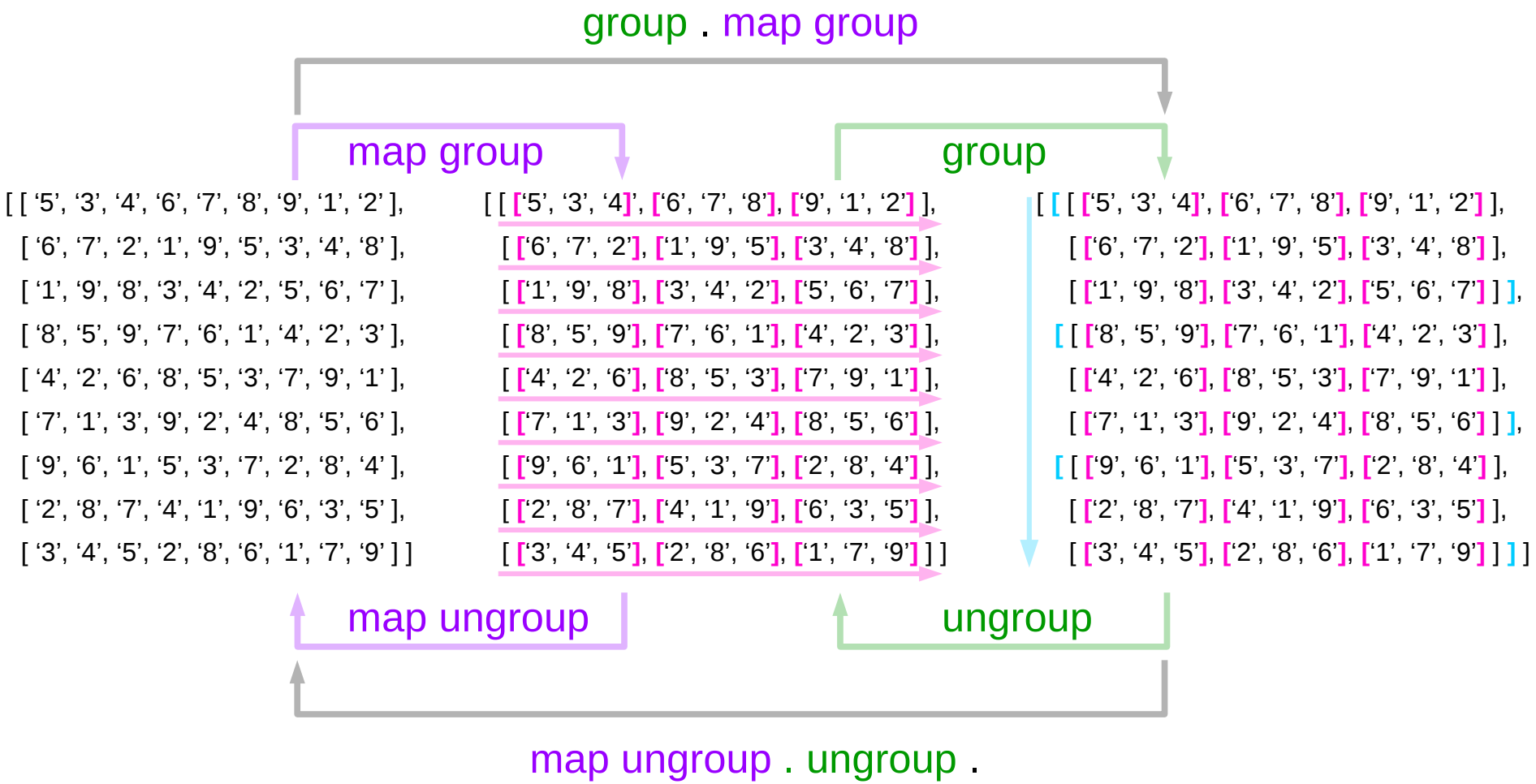


map group and map ungroup

```
map ungroup . ungroup .
group . map group
```

```
map ungroup . ungroup . group . map group
```

map ungroup . ungroup . group . map group



boxs.boxs = id

```
boxs  . boxs = id
```

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

```
map ungroup . ungroup . map cols . group . map group .
map ungroup . ungroup . map cols .  group . map group = id
```

Laws derived from boxs.boxs = id

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

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

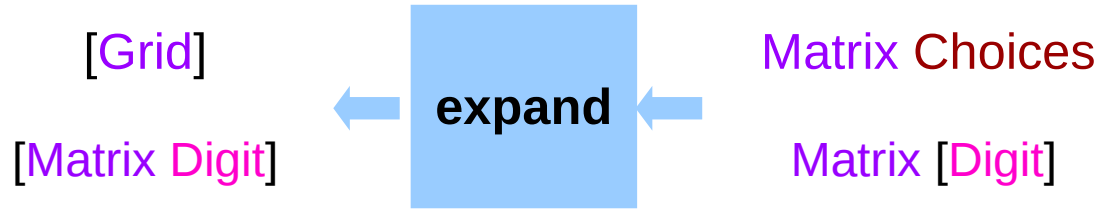
map ungroup . ungroup .
group . map group = id

Expand Laws

```
map rows . expand = expand . rows
map cols . expand = expand . cols
map boxs . expand = expand . boxs
```

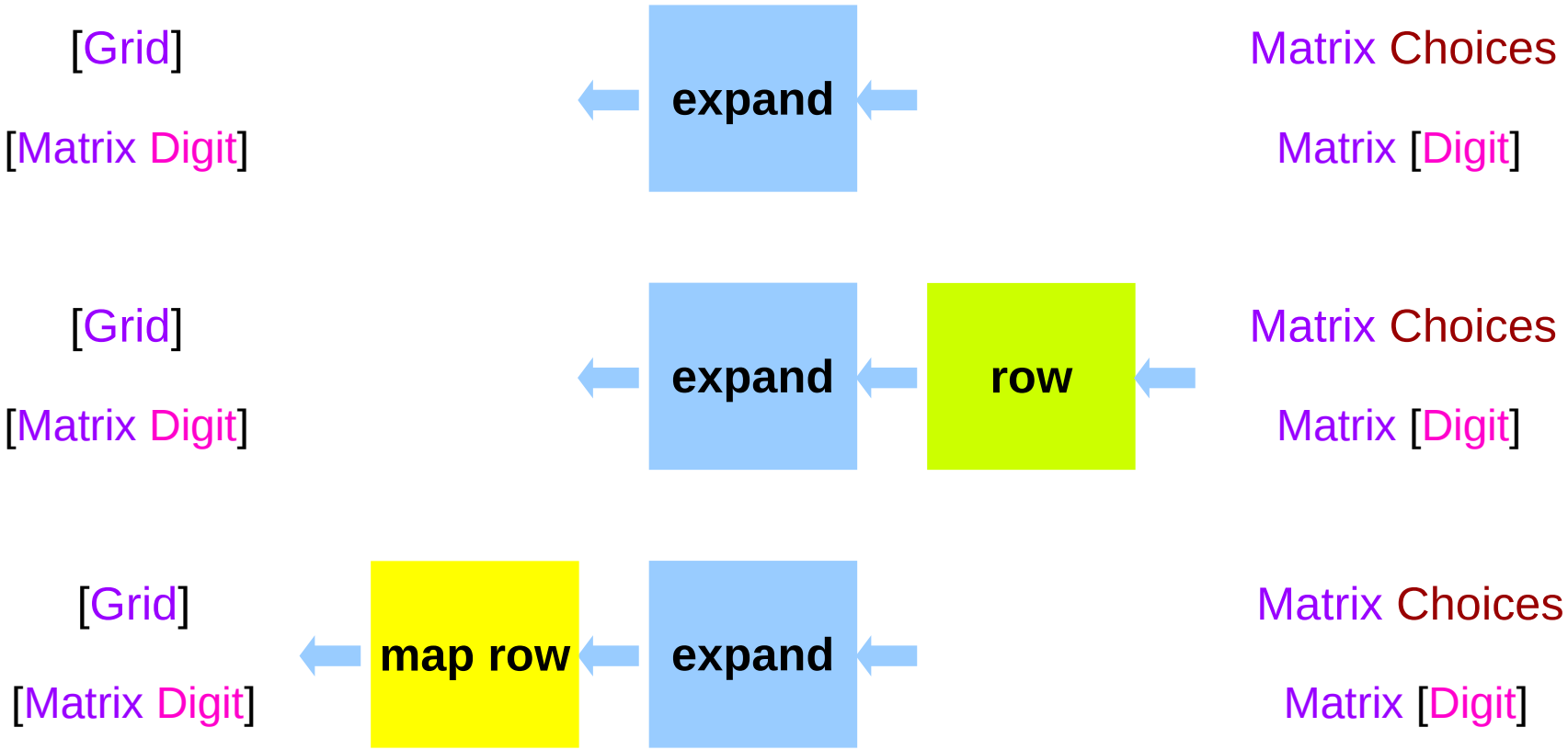
expand :: Matrix Choices -> [Grid]
expand = **cp** . map **cp**

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



expand . rows

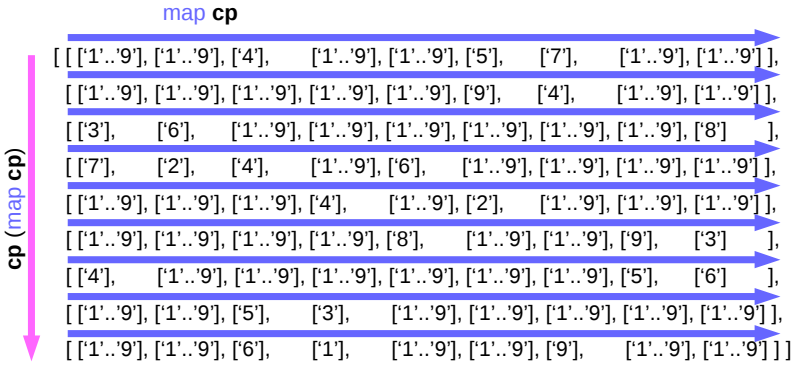
map **rows** . **expand** = **expand** . **rows**



expand – map cp

Matrix Choices

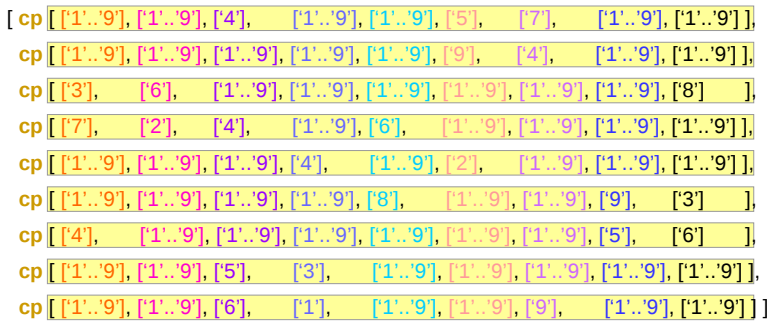
Matrix [Digit]

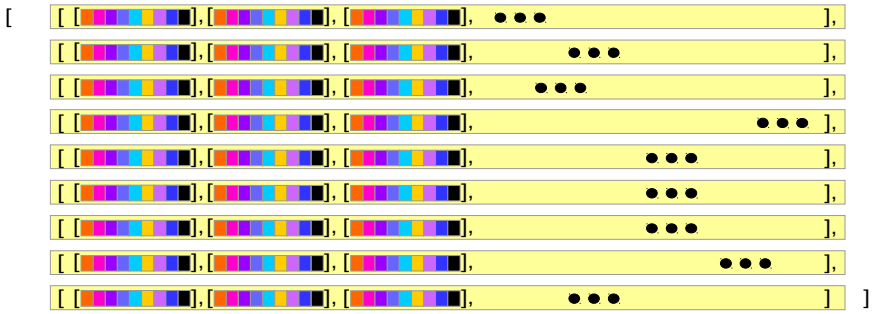


expand :: Matrix Choices -> [Grid]

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

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

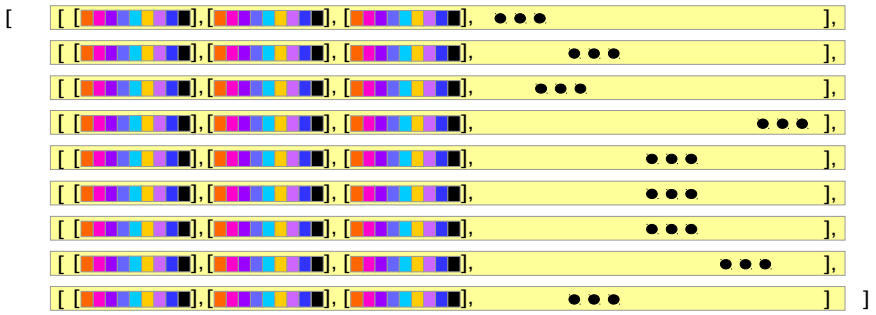




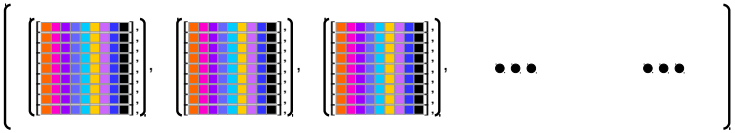
map **cp**

expand – cp . map cp

map **cp**

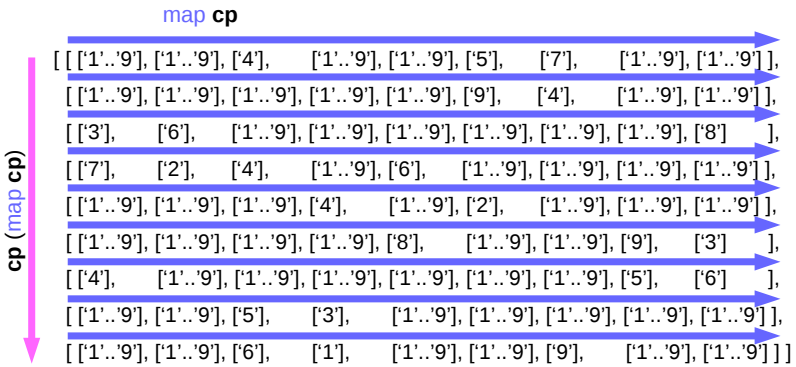


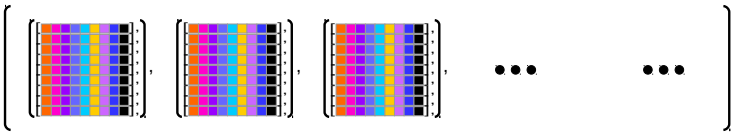
cp . map **cp**



[Grid]
[Matrix Digit]

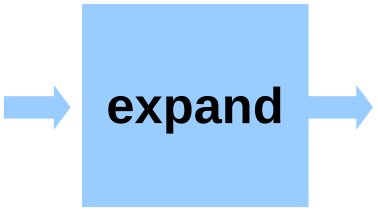
expand





Matrix Choices

Matrix [Digit]



[Grid]

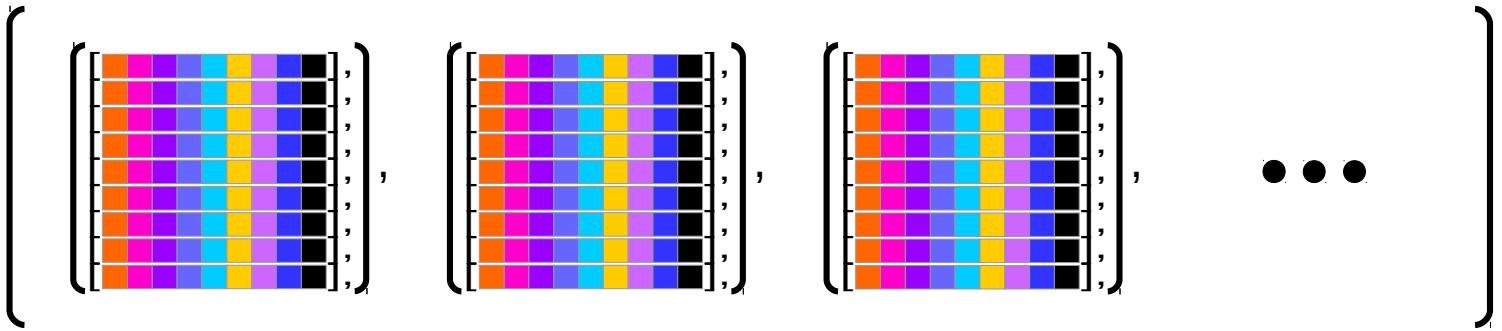
[Matrix Digit]

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

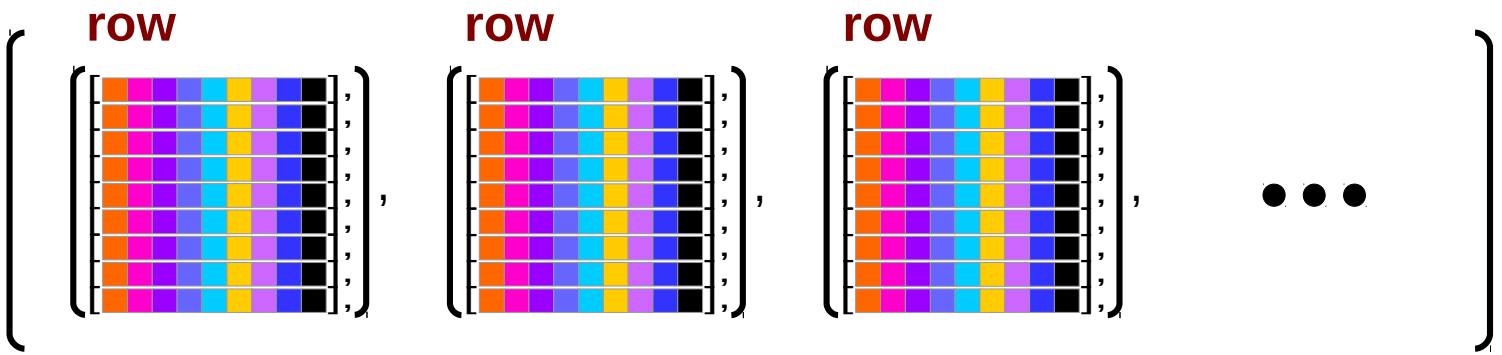
cp . map **cp** = [[[a]]] -> [[[a]]]

map row . expand

map row .



row **row** **row**



Cartesian Product (**cp**)

```
cp  ::     [[ a ]]   -> [[ a ]]
```

```
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** [] = [[ ]]

Double Map

```
f :: a -> b
map f :: [a] -> [b]
map (map f) :: [[a]] -> [[b]].
```

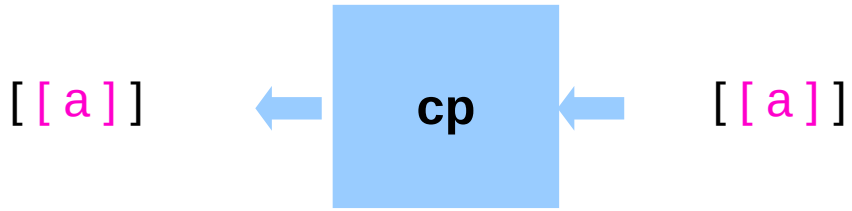
```
map           :: (a -> b) ->   [a]   ->   [b]
(map.map)     :: (a -> b) ->  [[a]]  ->  [[b]]
(map.map.map) :: (a -> b) -> [[[a]]] -> [[[b]]]
```

http://stackoverflow.com/questions/8735072/double-map-in-haskell

CP Laws

map (map **f**) . **cp** = **cp** . map (map **f**)
filter (all **p**) . **cp** = **cp** . map (filter **p**)

all **p** = and . map **p**



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

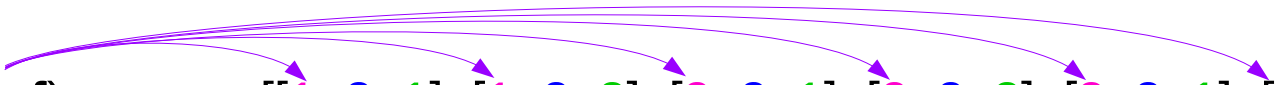
[[1, 2, 1], [1, 2, 3], [2, 2, 1], [2, 2, 3], [3, 2, 1], [3, 2, 3]]

Double map and cp (1)

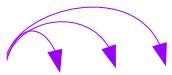
map (map **f**) . **cp** = **cp** . map (map **f**)

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

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



map (map **f**) [[1, 2, 1], [1, 2, 3], [2, 2, 1], [2, 2, 3], [3, 2, 1], [3, 2, 3]]



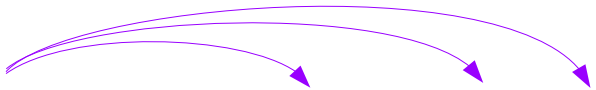
map (map **f**) **cp** A = [map **f** [1, 2, 1], map **f** [1, 2, 3], map **f** [2, 2, 1],
map **f** [2, 2, 3], map **f** [3, 2, 1], map **f** [3, 2, 3]]

= [[**f** 1, **f** 2, **f** 1], [**f** 1, **f** 2, **f** 3], [**f** 2, **f** 2, **f** 1],
[**f** 2, **f** 2, **f** 3], [**f** 3, **f** 2, **f** 1], [**f** 3, **f** 2, **f** 3]]

Double map and cp (2)

map (map **f**) . **cp** = **cp** . map (map **f**)

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



map (map **f**) [[1, 2, 3], [2], [1, 3]]

= [map **f** [1, 2, 3], map **f** [2], map **f** [1, 3]]

map (map **f**) A = [[**f** 1, **f** 2, **f** 3], [**f** 2], [**f** 1, **f** 3]]

cp . map (map **f**) A = [[**f** 1, **f** 2, **f** 1], [**f** 1, **f** 2, **f** 3], [**f** 2, **f** 2, **f** 1],
[**f** 2, **f** 2, **f** 3], [**f** 3, **f** 2, **f** 1], [**f** 3, **f** 2, **f** 3]]

Filter and cp

filter (all p) . cp = cp . map (filter p)
all p = and . map p

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

filter (all **p**) . **cp** A = filter [all **p** [1, 2, 1], all **p** [1, 2, 3], all **p** [2, 2, 1],
all **p** [2, 2, 3], all **p** [3, 2, 1], all **p** [3, 2, 3]]

= filter [and [**p** 1, **p** 2, **p** 1], and [**p** 1, **p** 2, **p** 3], and [**p** 2, **p** 2, **p** 1],
and [**p** 2, **p** 2, **p** 3], and [**p** 3, **p** 2, **p** 1], and [**p** 3, **p** 2, **p** 3]]

map (filter **p**) A = [filter **p** [1, 2, 3], filter **p** [2], filter **p** [1, 3]]
= [filter [**p** 1, **p** 2, **p** 3], [**p** 2], [**p** 1, **p** 3]]

cp . map (filter **p**) A = **cp** . [filter [**p** 1, **p** 2, **p** 3], [**p** 2], [**p** 1, **p** 3]]

References

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