ご注文は関数ですか?

akouryy

関数型プログラミング

関数型言語Haskell

- 値は不変(**変数**は存在しない)

全部定数

関数型言語Haskell

- 値は不変(**変数**は存在しない)
- 型がとても大事

コンパイル通ったら勝ち、ぬるぽ起きず

関数型言語Haskell

- 値は不変(**変数**は存在しない)
- 型がとても大事
- 配列より**片方向連結リスト**を多用

単方向–; これ以降単に「リスト」

関数型言語Haskell

- 値は不変(**変数**は存在しない)
- 型がとても大事
- 配列より**片方向連結リスト**を多用
- リスト操作が豊富(**高階関数**)

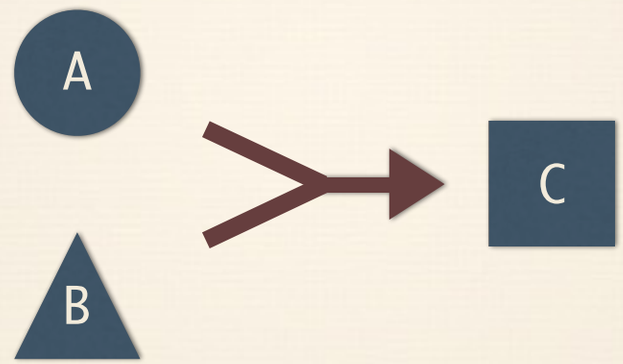
関数を引数にとる関数

Haskellの関数

例を2, 3個

カリー化

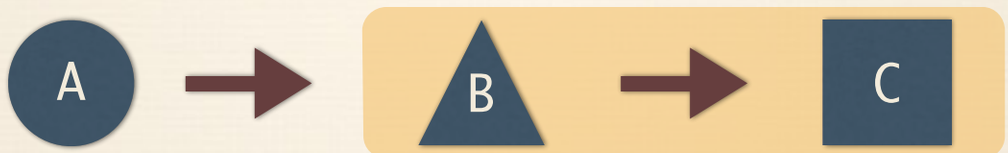
- 複数の引数をとる関数



AとBをとってCを返す関数に見える

カリー化

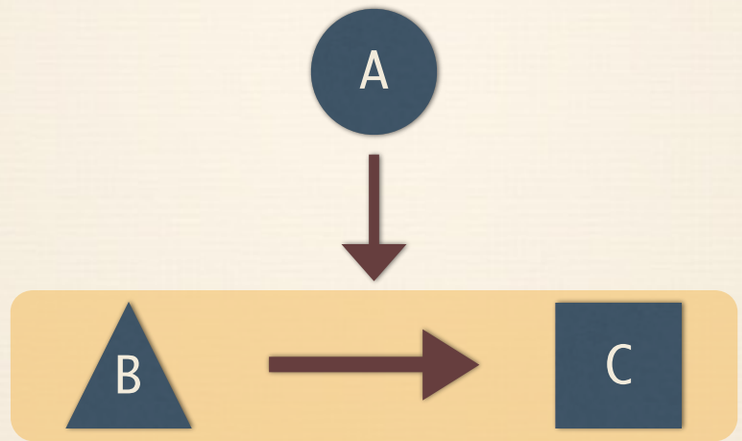
- 複数の引数をとる関数



実態はAをとって「BをとってCを返す関数」を返す関数

カリー化

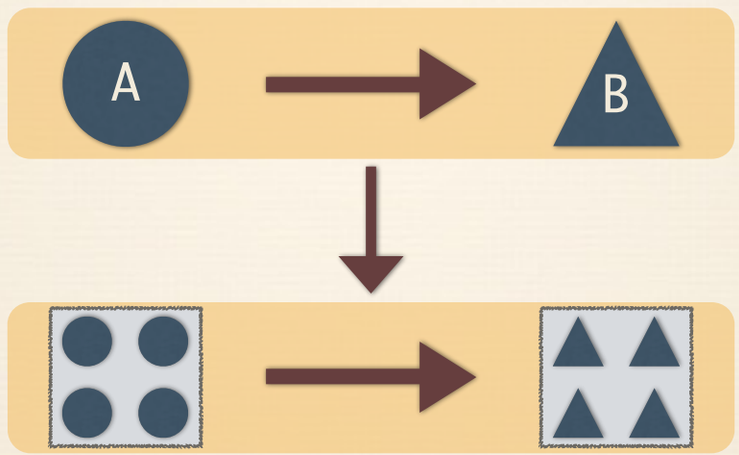
- 複数の引数をとる関数



以降の図でこの形があったらカリー化できるということ

map関数

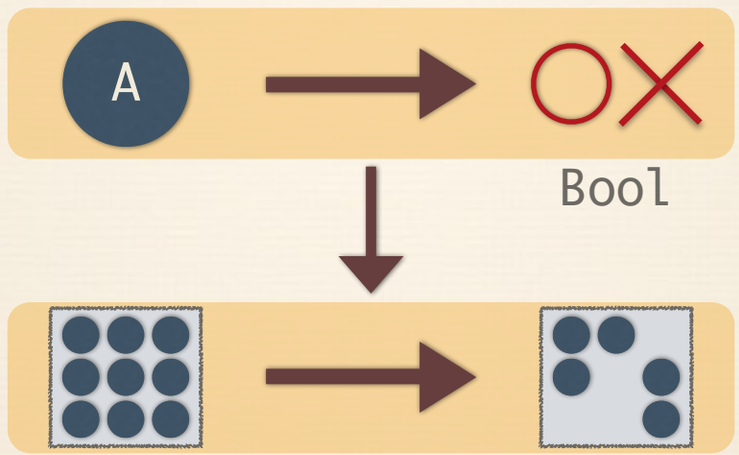
- 関数をリストの各要素に適用する関数



それぞれの要素に適用するときfor等を使わずにmap関数で一発変換

filter関数

- 述語を満たす要素のみを集める関数



要素を取捨選択するのはfilter関数で一発

数独

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

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

選択肢を列挙

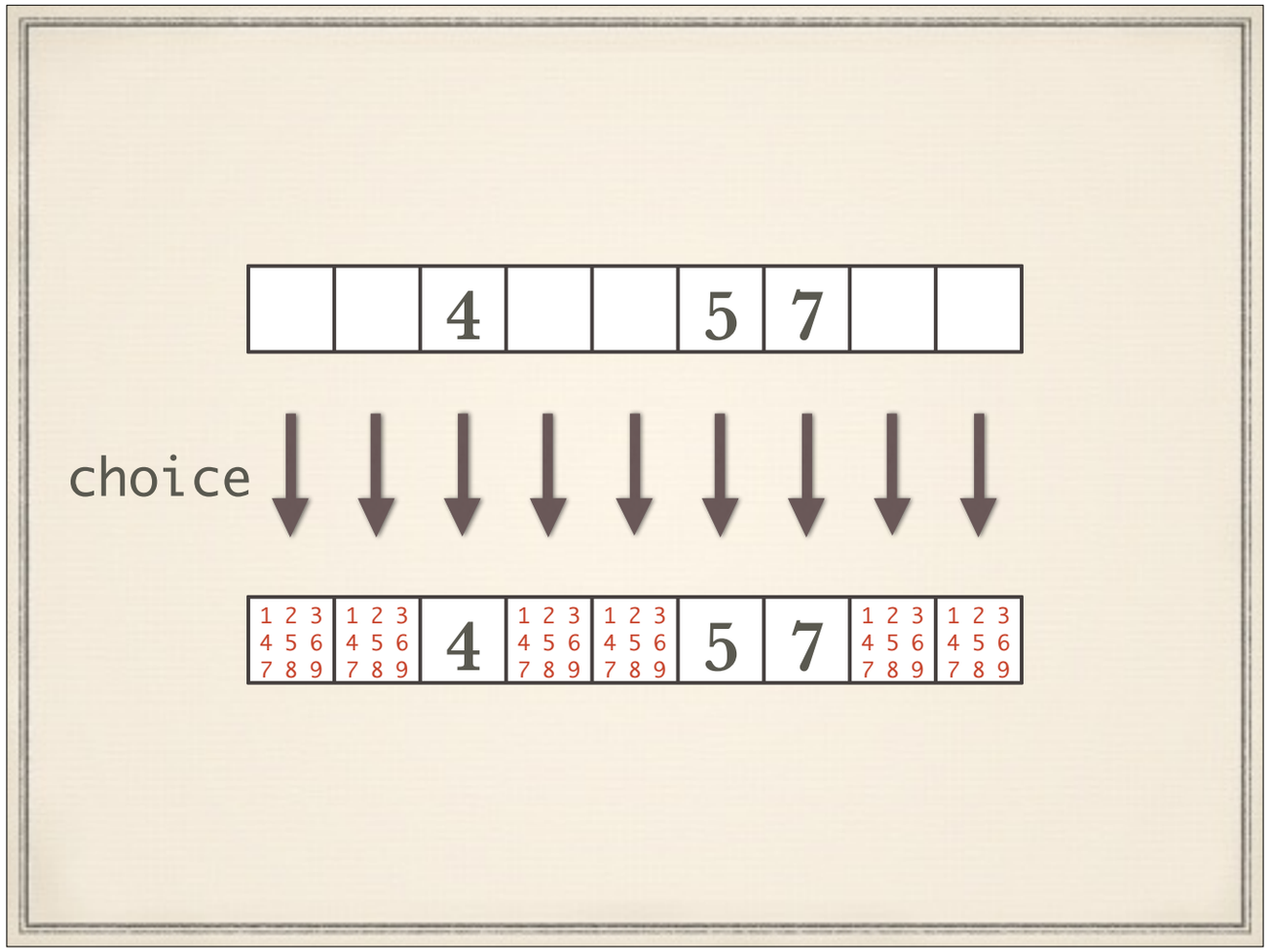
1マスの選択肢

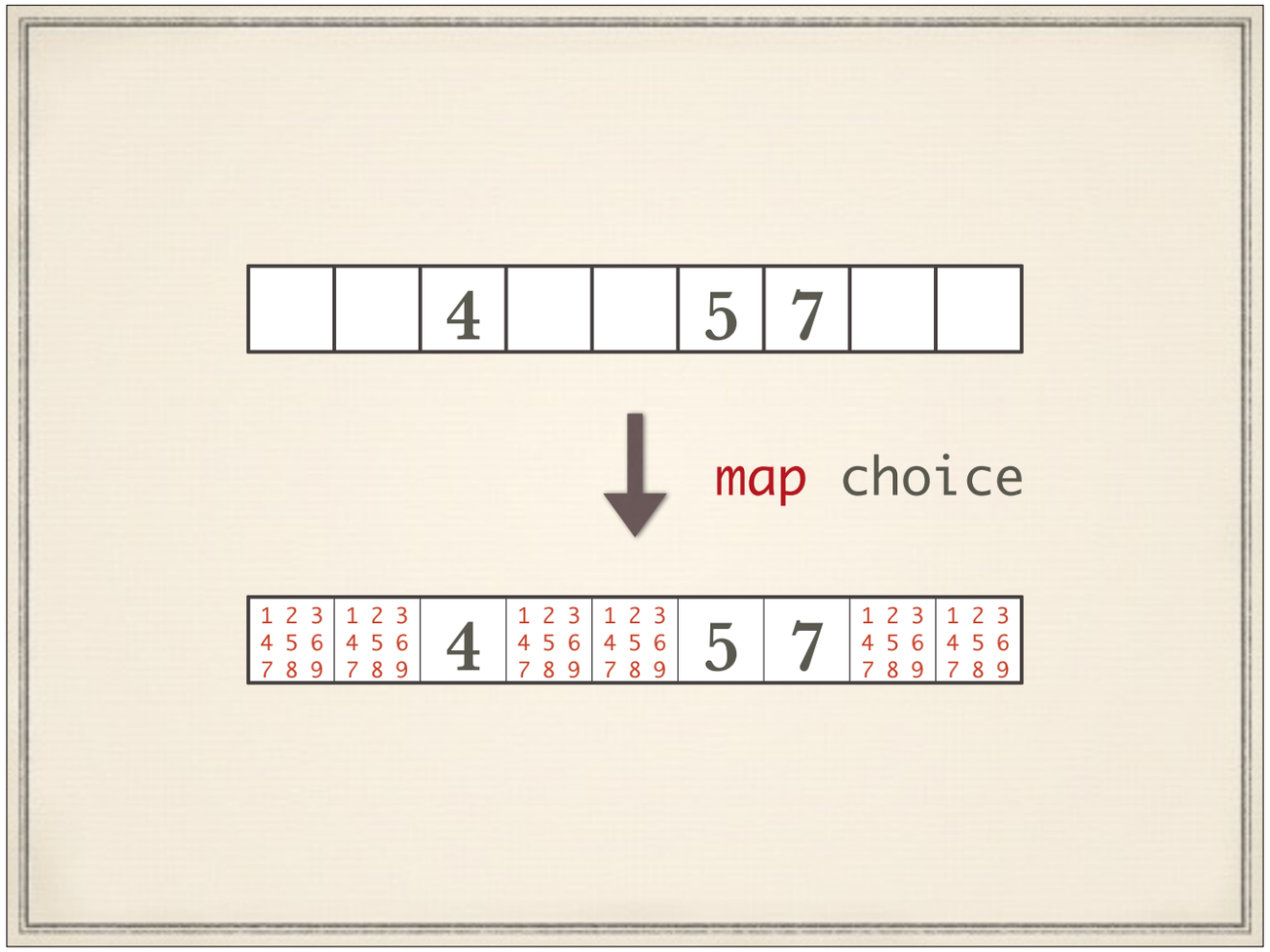
- 0 → 1〜9の9通り

choice

- 1〜9 → その数字で固定

		4			5	7		


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


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

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

map (map choice)

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

直積

- ここで リストの直積(cp)を考える

```
cp [[1, 2], [3], [4, 5, 6]]
== [[1, 3, 4], [1, 3, 5], [1, 3, 6],
    [2, 3, 4], [2, 3, 5], [2, 3, 6]]
```

盤面全体の選択肢

- 盤面全体の選択肢はchoicesの直積

```
map (map cp)
```

1	1	4
1	1	1
3	6	1

1	1	4
1	1	1
3	6	2

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

1	1	4
1	1	1
3	6	3

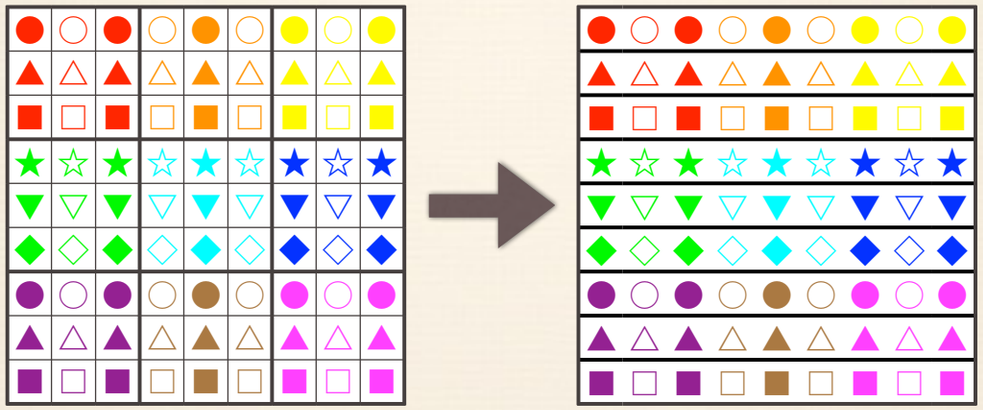
1	1	4
1	1	1
3	6	4

9	9	4
9	9	9
3	6	9

正解判定

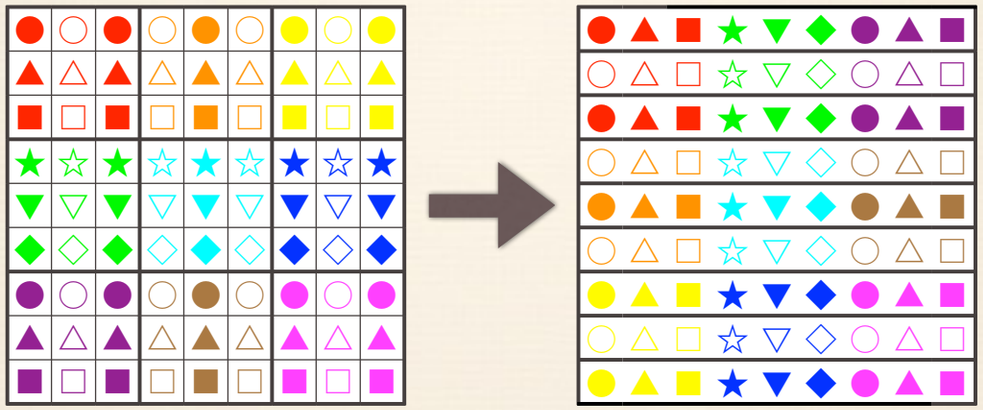
行、列、ブロック

行



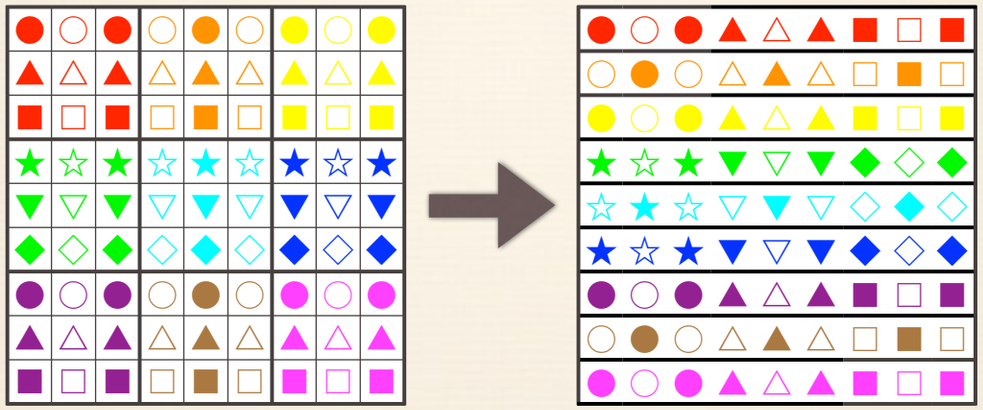
行、列、ブロック

列



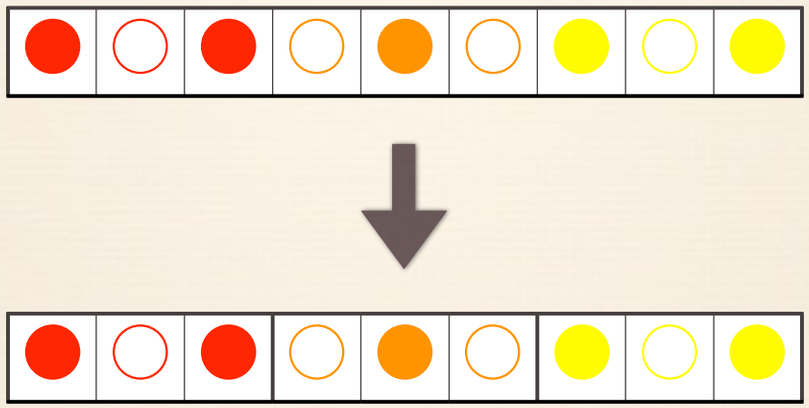
行、列、ブロック

ブロック



行、列、ブロック

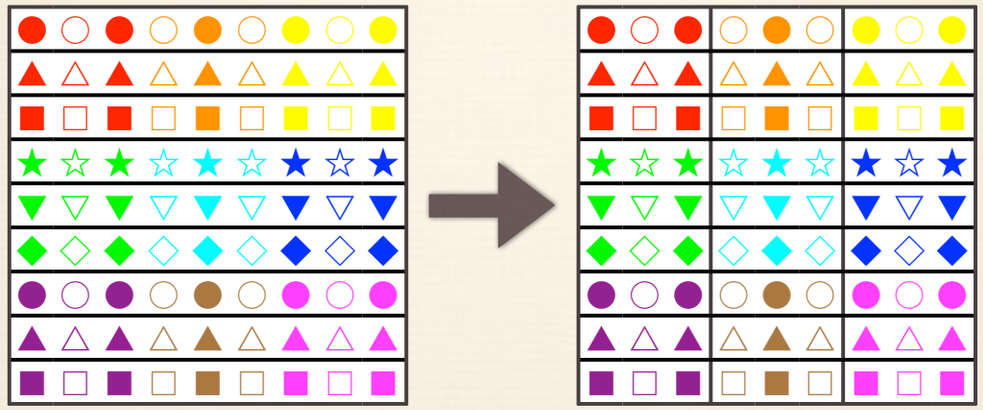
ブロック



をmapする

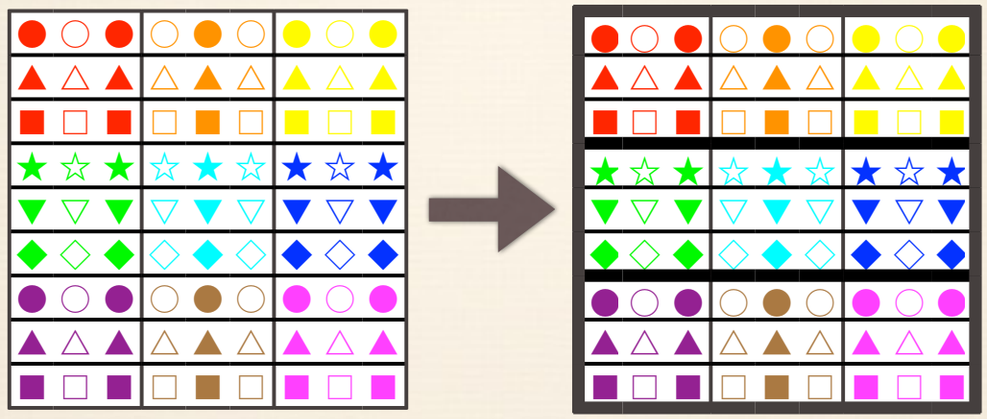
行、列、ブロック

ブロック



行、列、ブロック

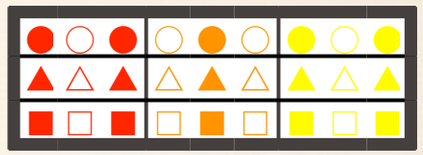
ブロック



なぜ分けたかというと

行、列、ブロック

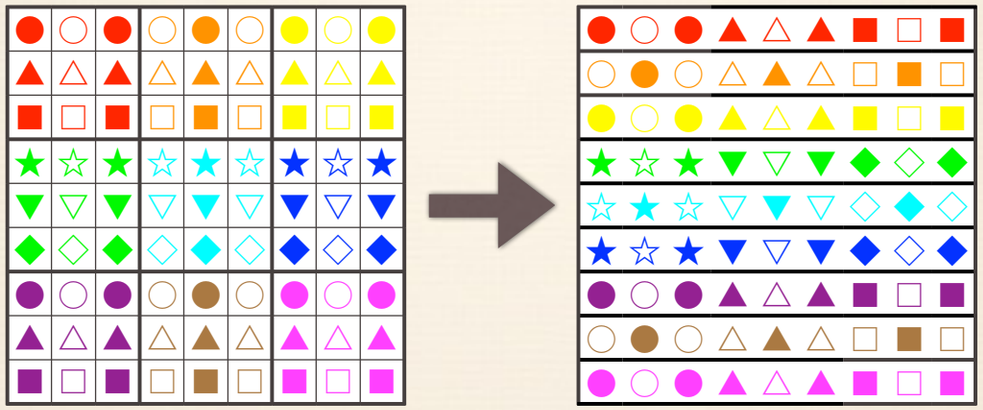
ブロック



この3列を一まとめにして関数適用したいから

行、列、ブロック

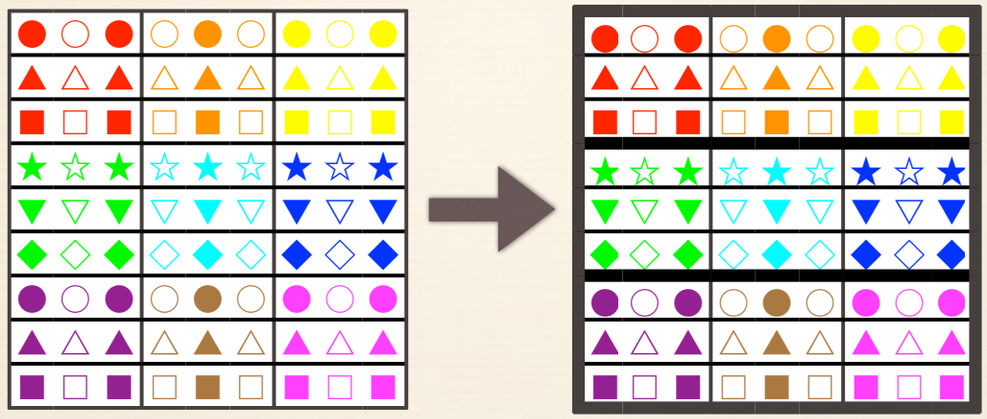
ブロック



目標

行、列、ブロック

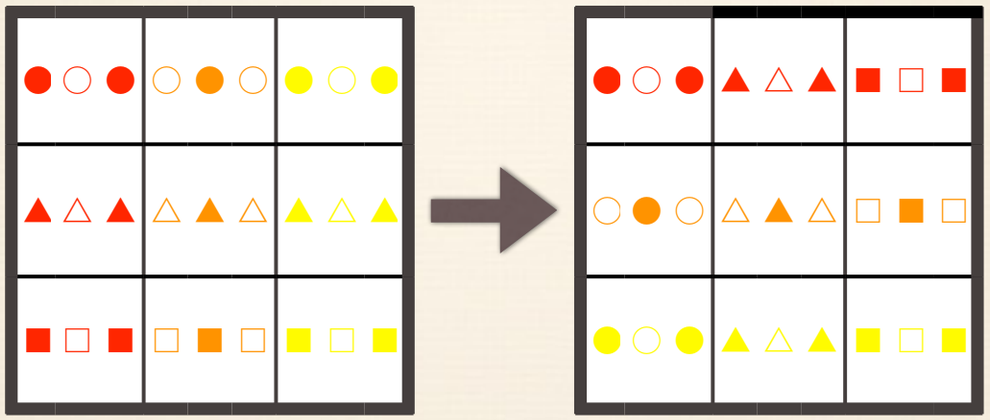
ブロック



なぜ分けたかというと

行、列、ブロック

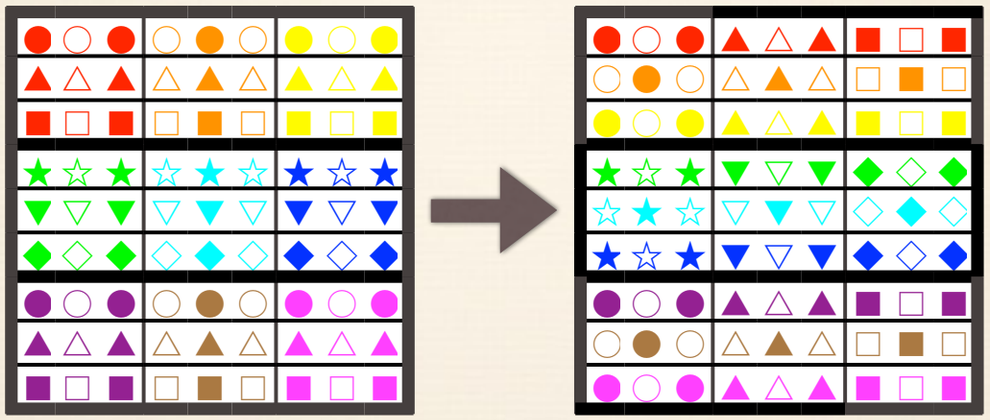
ブロック



列のときに定義した縦横変換

行、列、ブロック

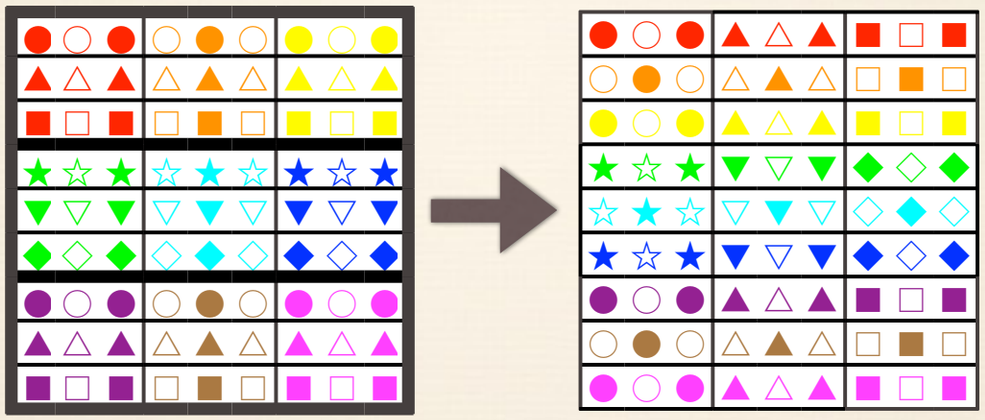
ブロック



mapする

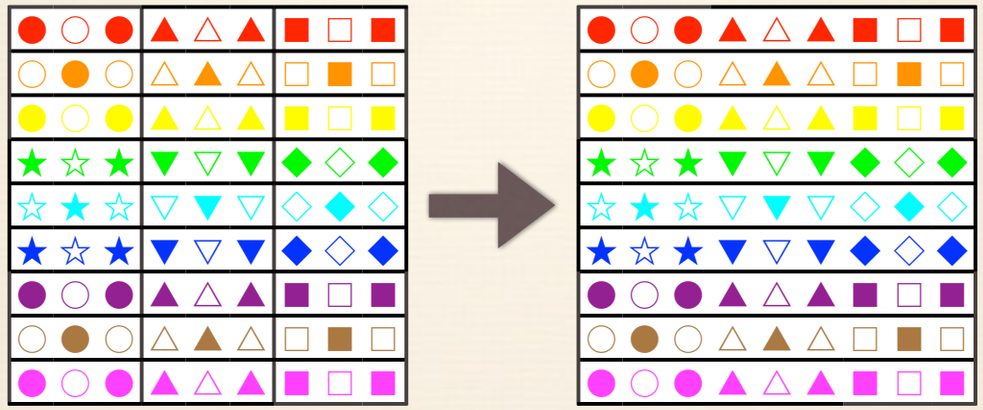
行、列、ブロック

ブロック



行、列、ブロック

ブロック



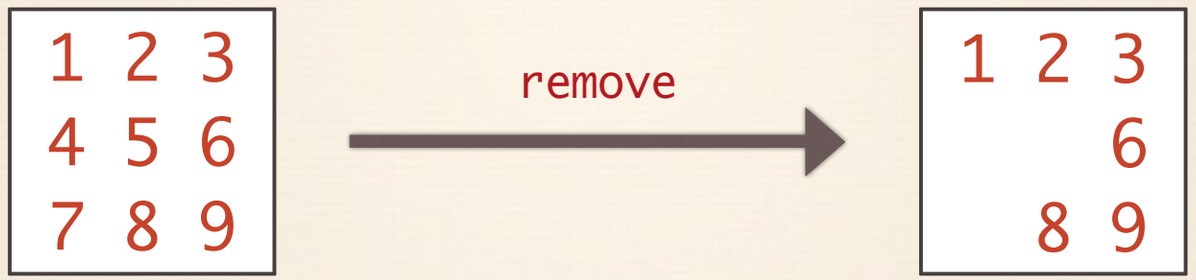
候補の絞り込み

絞り込み

- 全探索 → $O(n^{n^2})$
- 重複を除いていく

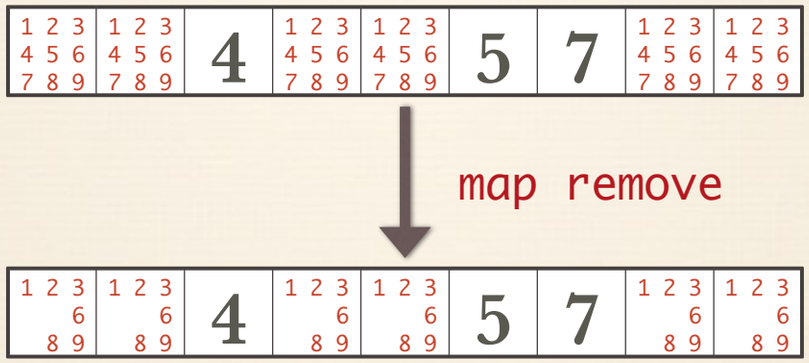
行の重複排除

- 同じ列に現れる数を候補から消す



行の重複排除

- 同じ列に現れる数を候補から消す



重複排除

- 列、ブロックはcolsやboxsで変換、逆変換

絞り込み

- 重複排除を繰り返す

絞り込み

- 重複排除を繰り返す
- 候補の数がm個に絞り込めた → $O(mn^3+n^6)$

頑張れば5乗

絞り込み

- 重複排除を繰り返す
- 候補の数がm個に絞り込めた → $O(mn^3+n^6)$
- mが小さいとき全探索$O(n^{n^2})$から圧倒的改善

簡単じゃないとmが大きくてダメ

候補の列挙

候補の列挙

- これまでは候補を全て同時に列挙していた

候補の列挙

- これまでは候補を全て同時に列挙していた
- 1マスごとに候補列挙→絞り込みを繰り返す

候補の列挙

- 候補が少ないマスから列挙していく

あまり面白くないので省略

候補の列挙

- 候補が少ないマスから列挙していく
- $O(mn^3+n^5) \rightarrow O(n^6)$

まとめ

Haskellで書くと、データを更新せず、
新たなデータを操作でどんどん作っていく
手順ではなく操作に集中
みなさんもHaskellを使いましょう

ソースコード

joiss.ぴょんぴょん.net

ご注文はHaskellですか?

akouryy