SOLVING SUDOKU

Graham Hutton
University of Nottingham

(with thanks to Richard Bird)

What is Sudoku?

- A simple but addictive puzzle, invented in the USA in 1979 and called Number Place;
- Became popular in Japan in 1986, where it was renamed Sudoku (~ “single number”);
- First appeared in UK newspapers in 2004, and became an international craze in 2005.

Example

Fill in the grid so that every row, column and box contains each of the numbers 1 to 9:

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

Example

Fill in the grid so that every row, column and box contains each of the numbers 1 to 9:

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

What number must go here?

Example

Fill in the grid so that every row, column and box contains each of the numbers 1 to 9:

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

1, as 2 and 3 already appear in this column.

Example

Fill in the grid so that every row, column and box contains each of the numbers 1 to 9:

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

Example

Fill in the grid so that every row, column and box contains each of the numbers 1 to 9:

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

Example

Fill in the grid so that every row, column and box contains each of the numbers 1 to 9:

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

Example

Fill in the grid so that every row, column and box contains each of the numbers 1 to 9:

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

Example

Fill in the grid so that every row, column and box contains each of the numbers 1 to 9:

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

And so on...

Example

Fill in the grid so that every row, column and box contains each of the numbers 1 to 9:

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

The <u>unique</u> solution for this <u>easy</u> puzzle.

This Talk

- We show how to develop a program that can solve any Sudoku puzzle in an instant;

- Start with a simple but impractical program, which is improved in a series of steps;

- Emphasis on pictures rather than code, plus some lessons about algorithm design.

Representing a Grid

```
type Grid       =  Matrix Char

type Matrix a   =  [Row a]

type Row a      =  [a]
```

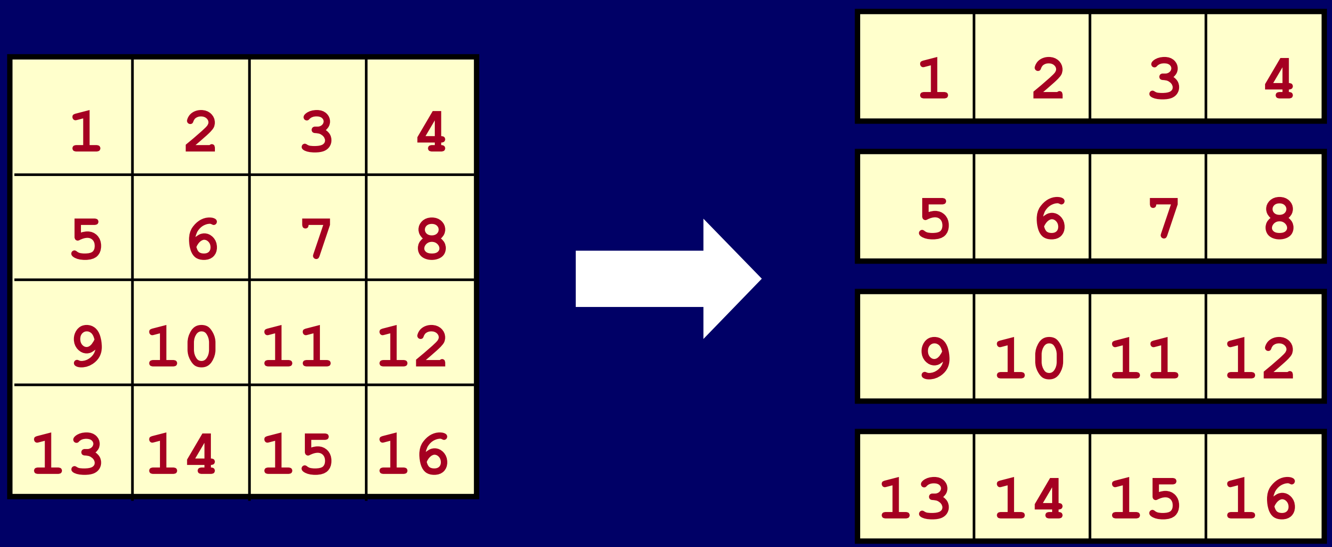
A grid is essentially a list of lists, but matrices and rows will be useful later on.

Examples

```
empty :: Grid
empty  = replicate 9 (replicate 9 ' ')
```

```
easy  :: Grid
easy   = [ "2    1 38" ,
           "        5" ,
           " 7   6   " ,
           " 13      " ,
           " 981  257" ,
           "31    8  " ,
           "9  8   2 " ,
           " 5  69784" ,
           "4 25     " ]
```

Extracting Rows




```
rows   :: Matrix a → [Row a]
rows m = m
```

… Columns

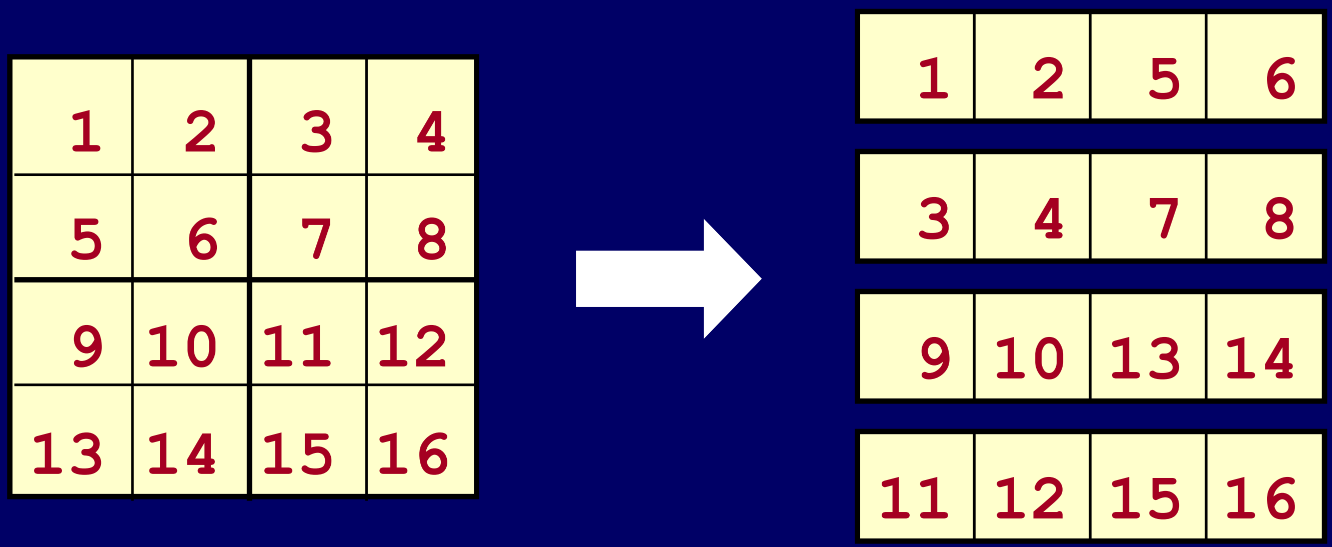
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

$\Rightarrow$

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

```
cols   :: Matrix a → [Row a]
cols m = transpose m
```

… And Boxes




```
boxs   :: Matrix a → [Row a]
boxs m = <omitted>
```

Validity Checking

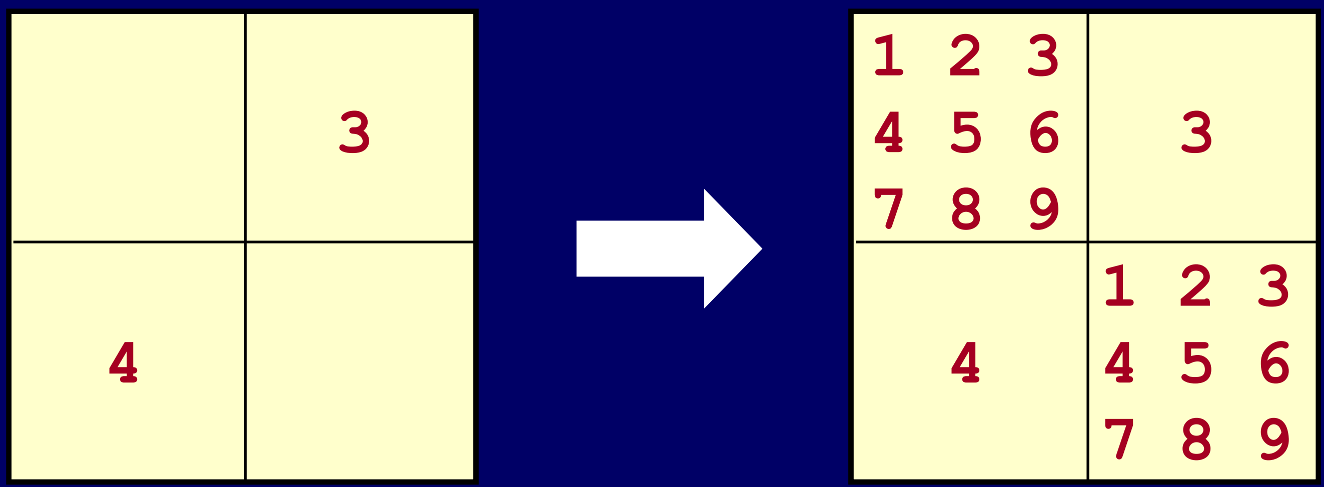
Let us say that a grid is <u>valid</u> if it has no duplicate entries in any row, column or box:

```
valid   :: Grid → Bool
valid g = all nodups (rows g) ∧
          all nodups (cols g) ∧
          all nodups (boxs g)
```

A direct implementation, without concern for efficiency.

Making Choices

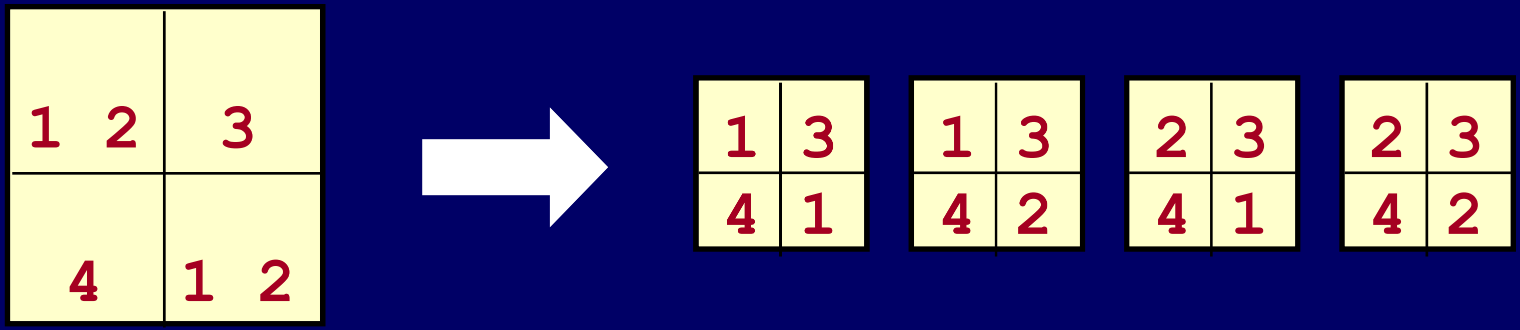
Replace each blank square in a grid by all possible numbers 1 to 9 for that square:




```
choices :: Grid → Matrix [Char]
```

Collapsing Choices

Transform a matrix of lists into a list of matrices by considering all combinations of choices:




```
collapse :: Matrix [a] → [Matrix a]
```

A Brute Force Solver

```
solve :: Grid → [Grid]
solve  = filter valid . collapse . choices
```

Consider all possible choices for each blank square, collapse the resulting matrix, then filter out the valid grids.

Does It Work?

The easy example has 51 blank squares, resulting in 9^{51} grids to consider, which is a huge number:

4638397686588101979328150167890591454318967698009

```
> solve easy
ERROR: out of memory
```

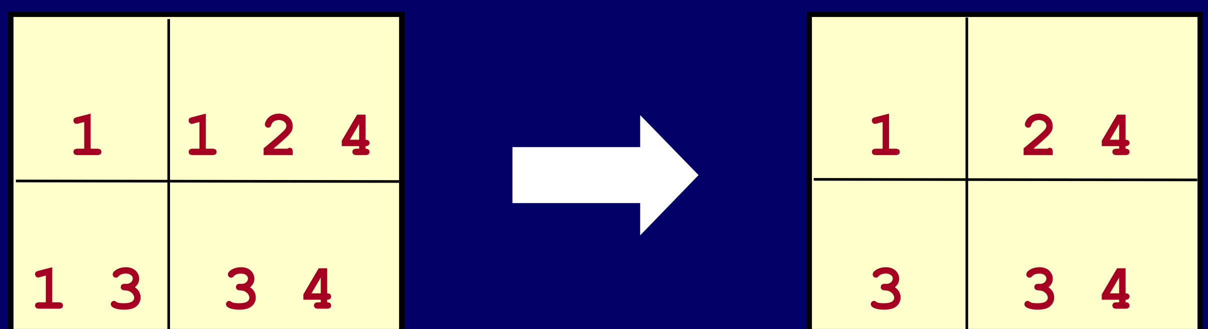
Simple, but impractical!

Reducing The Search Space

- Many choices that are considered will <u>conflict</u> with entries provided in the initial grid;
- For example, an initial entry of 1 <u>precludes</u> another 1 in the same row, column or box;
- <u>Pruning</u> such invalid choices before collapsing will considerably reduce the search space.

Pruning

Remove all choices that occur as single entries in the corresponding row, column or box:




```
prune :: Matrix [Char] → Matrix [Char]
```

And Again

Pruning may leave new single entries, so it makes sense to iterate the pruning process:

1	2 4
3	3 4

And Again

Pruning may leave new single entries, so it makes sense to <u>iterate</u> the pruning process:

1	2 4
3	4

And Again

Pruning may leave new single entries, so it makes sense to <u>iterate</u> the pruning process:

1	2
3	4

And Again

Pruning may leave new single entries, so it makes sense to iterate the pruning process:

1	2
3	4

We have now reached a fixpoint of the pruning function.

An Improved Solver

```
solve' :: Grid → [Grid]
solve'  =
  filter valid . collapse . fix prune . choices
```

For the easy example, the pruning process alone is enough to completely solve the puzzle:

```
> solve' easy
```

Terminates instantly!

But…

For a gentle example, pruning leaves around 3^{81} grids to consider, which is still a huge number:

443426488243037769948249630619149892803

```
> solve' gentle
```

No solution after two hours - we need to think further!

Reducing The Search Space

- After pruning there may still be many choices that can never lead to a solution;
- But such bad choices will be duplicated many times during the collapsing process;
- Discarding these bad choices is the key to obtaining an efficient Sudoku solver.

Blocked Matrices

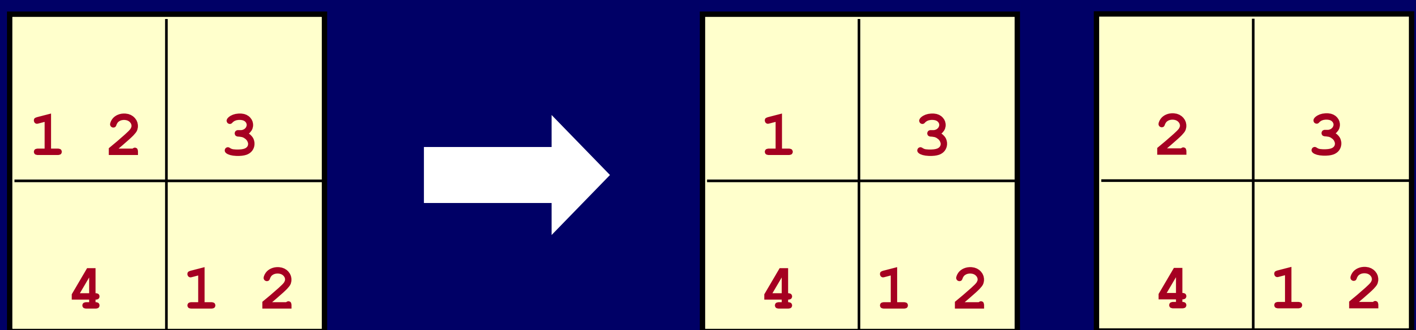
Let us say that a matrix is blocked if some square has no choices left, or if some row, column, or box has a duplicated single choice:

1	1 2	1
3 4		3

Key idea - a blocked matrix can never lead to a solution.

Expanding One Choice

Transform a matrix of lists into a list of matrices by expanding the <u>first</u> square with choices:




```
expand :: Matrix [a] → [Matrix [a]]
```

Our Final Solver

```
solve'' :: Grid → [Grid]
solve''  = search . prune . choices
```

```
search :: Matrix [Char] → [Grid]
search m
   | blocked m  = []
   | complete m = collapse m
   | otherwise  = [g | m' ← expand m
                     , g  ← search (prune m')]
```

Notes

- Using fix prune rather than prune makes the program run <u>slower</u> in this case;

- No need to filter out valid grids, because they are guaranteed to be valid by <u>construction</u>;

- This program can now solve any newspaper Sudoku puzzle in an <u>instant</u>!

The Result

This program has saved my life - my Sudoku addiction is finally cured!!

Subliminal Message

Haskell is the world's greatest programming language.