Université de Liège

Logics for Computer Science

INFO9015-1

Solving Sudokus

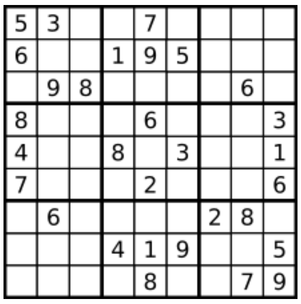
Auteure :

Navez Lucie (s180703)

Master ingénieur civil

Année académique 2022-2023

1 Formalism and statement



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

Before doing anything it is important to fix the ideas about how sudokus work and find the logical statements that can be derived from that.
The Sudoku grids have the following characteristics :

- The Sudoku grid has a 9x9 shape
- The grid is divided into blocks of length 3, thus, the global grid is divided into 9 sub-grids.
- Numbers ranging from 1 to 9 (included) are already present in the grid at the beginning of the game.

The rules are the following : Each number must be present maximum :

- Once in a row of the global grid
- Once in a column of the global grid
- Once in a 3x3 block of the grid

These rules can help to form the logical statements.

1.1 Generalization

In a more general way, sudokus can also be viewed as $\mathbf{N} * \mathbf{N}$ grids, where the blocks are units of size $\sqrt{\mathbf{N}} * \sqrt{\mathbf{N}}$. The grid is partially filled with values going from 1 to $\mathbf{N}$, and the rules remain the same : each column, each row and each block must contain one occurrence of each number from 1 to $\mathbf{N}$.

1.2 Rules

We will denote each cell by : its row (r), its column (c) and the digit assigned to it (d).
Thus, each cell will be characterized as follows : (r, c, d).
We need to define propositions to write formulas. Each cell can either be filled, or not, then, the propositions we will use will be defined as follows :

$$p_{r,c,d} = 1 \text{ if filled}, 0 \text{ otherwise}$$

Thus, the following statements can be translated into logical statements :

- Each cell must be filled with a number :
  For each row, and for each column, each cell must contain a value between 1 and $\mathbf{N}$:

$$\bigwedge_r \bigwedge_c \bigvee_d p_{r,c,d}$$

- Each digit must appear only once in a row : For each column, and each possible value, cells from the same row cannot simultaneously contain the same value.

$$\bigwedge_c \bigwedge_d \bigwedge_{r-1} \bigwedge_{r'=r+1}^{N} (\neg p_{r,c,d} \vee \neg p_{r',c,d})$$

- Each digit must appear only once in a column : For each row, and each possible value, cells from the same column cannot simultaneously contain the same value.

$$\bigwedge_r \bigwedge_d \bigwedge_{c-1} \bigwedge_{c'=c+1}^{N} (\neg p_{r,c,d} \vee \neg p_{r,c',d})$$

- Each digit must appear only once in a $\sqrt{\mathbf{N}} \times \sqrt{\mathbf{N}}$ block :

$$\bigwedge_c \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{r=1}^{\sqrt{\mathbf{N}}} \bigwedge_{c=1}^{\sqrt{\mathbf{N}}} \bigwedge_{c'=c+1}^{\sqrt{\mathbf{N}}} (\neg p_{(3i+r),(3j+c),d} \vee \neg p_{(3i+r),(3j+c'),d})$$
$$\bigwedge_c \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{r=1}^{\sqrt{\mathbf{N}}} \bigwedge_{c=1}^{\sqrt{\mathbf{N}}} \bigwedge_{r'=r+1}^{\sqrt{\mathbf{N}}} \bigwedge_{k=1}^{\sqrt{\mathbf{N}}} (\neg p_{(3i+r),(3j+c),d} \vee \neg p_{(3i+r'),(3j+k),d})$$

2 Functionalities

The presented script implements the following functionalities :

Solve 9x9 sudokus The previous logical statements are encoded in SAT by using values ranging from 1 to 9, for instance, the following statement : cell (1, 1) contains value 5 is encoded in SAT as : 225. NB : Please note that since the SAT solver cannot use 0, the rows and columns (usually ranging from 0 to N-1, will be ranging from 1 to N, then corrected so that it fits the Python standards).

Solve 16x16 and 25x25 sudokus The reasoning is the same for bigger grids, but a problem arises. If one uses the same encoding as for 9x9 grids, some ambiguities might appear, for instance the SAT proposition 1215 can be interpreted in several ways :
- Cell (0, 1) contains value 15
- Cell (11, 0) contains value 5
- Cell (0, 20) contains value 5

To avoid such ambiguities, a solution is to simply ensure that all variables (row, column and value) are encoded on the same number of digits : 2, for instance. To do so, we simply encode the proposition in SAT by adding 10 to their real value, which yields, for instance : 111225 : Cell (0, 1) contains value 15, which obtain without any ambiguity.

Determine if a solution is unique To determine if a solution is unique, the SAT solver must try to find for a solution that is different from the previously found one (that is, at least one cell must be different from the previously found solution). These are actually constraints that can be added to the SAT problem. If the SAT solver is able to find a new solution with these new constraints, then the solution is not unique. To test this feature, a function able to generate sudokus with non-unique solutions quickly was implemented in `sudoku_utils.py`

Create 9x9 sudokus To create new sudokus, one uses a backtracking algorithm. This algorithm is implemented in two steps :

(1) First, generate a complete grid by using backtracking :
    Repeat the following as long as some cells of the grid are free :
    i. Choose a random cell and compute the values that could fill it according to the game rules.
    ii. Fill the cell with the number
    iii. Solve the sudoku
        - If the sudoku is not solvable, undo the previous steps and try again with another cell
        - If the sudoku is solvable, keep going

(2) Second, Remove some values from the sudoku one by one, and check if it still has a solution.

To create bigger sudokus (16x16 and 25x25), the process is the same. Note that it can take some time, especially for the 25x25 grids, because of the backtracking algorithm.

To create a sudoku that has a limited number of clues initially, the procedure can be repeated as long as the generated sudoku has more clues than the expected number, but this can be very long.

User interface for the previous functionalities The following functionalities can be tested in the graphical interface defined in `sudokuUI.py`. The following command line must be used to launch the interface :

```
python3 sudokuUI.py
```