Sudoku Solver, an Expert System

ATIKUR RAHMAN : 16701016
Department of **Computer Science and Engineering, University of Chittagong**

Problem Statement

Sudoku, originally called **Number Place** is a **9×9** grid puzzle based on **logic**, **combinatorial** number placement problem. In the grid, each box contains a number in the range 1-9 satisfying that each column, each row and each **3×3** block (i.e. the sub-blocks of the original **9×9** grid) contains all the numbers from 1-9 only once.

Solving Sudoku has been a challenging problem in the last decade. The purpose has been to develop more effective algorithm in order to reduce the computing time and utilize lower memory space.

Aims and Objectives

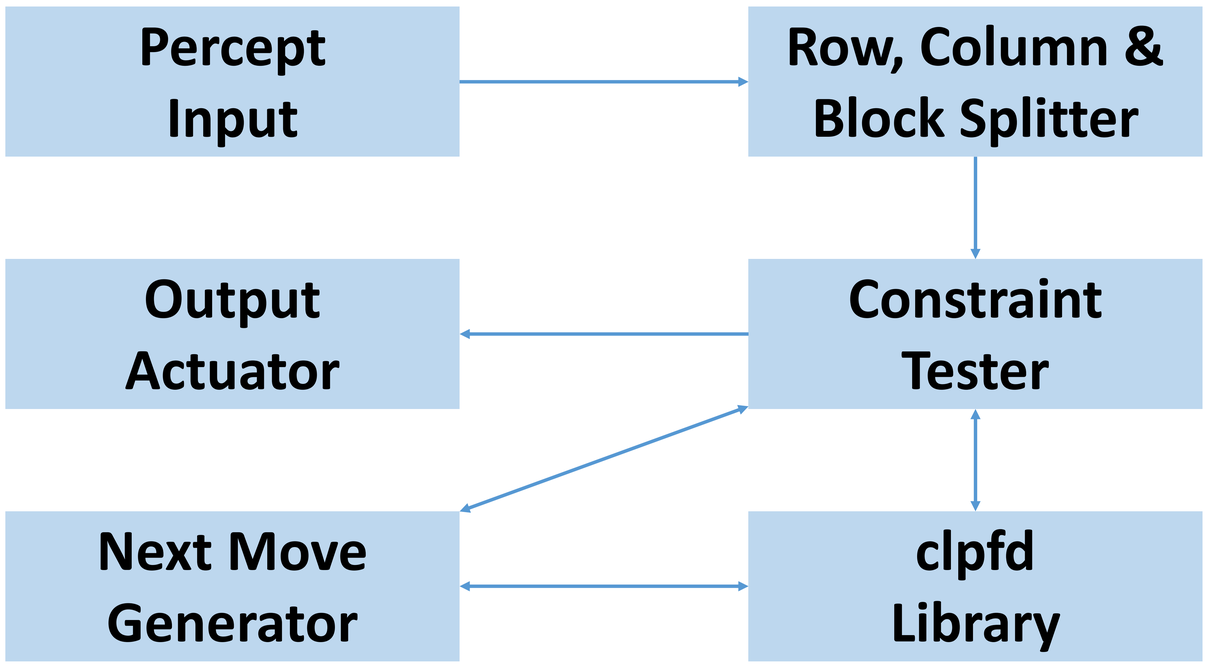
The general problem of solving Sudoku puzzles on $n^2 \times n^2$ grids of $n \times n$ blocks is known to be NP-Complete. Many computer algorithms, such as backtracking and dancing-links can solve most 9×9 puzzles efficiently, but combinatorial explosion occurs as n increases, creating limits to the properties of Sudokus that can be constructed, analyzed, and solved as n increases. A Sudoku puzzle can be expressed as a graph-coloring problem. The aim is to construct a 9-coloring of a particular graph, given a partial 9-coloring.

The aim of this project is to convert the puzzle to a **Constraints Satisfaction Problem (CSP).** Then implement it in **Prolog** environment to build an expert system which will be able to solve a given sudoku puzzle.

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

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

Design



Screenshot of Execution Protocol

```
✔=> /home/chitholian
chitholian@ChitholianLinux 11:50:42 AM 13% $ swipl -f sudoku.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.0.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- Puzzle = [
    [2,_,_,_,9,_,_,_,1],
    [3,_,9,_,_,7,_,_,_],
    [_,_,1,_,4,_,_,7,_],
    [_,6,_,_,_,_,_,_,_],
    [_,_,_,_,_,3,_,_,_],
    [_,_,8,6,_,_,7,9,_],
    [6,_,_,7,_,_,8,_,_],
    [1,2,3,_,_,8,_,_,_],
    [_,8,7,_,_,4,3,_,_]], Puzzle = [A,B,C,D,E,F,G,H,I], sudoku(Puzzle).
Puzzle = [[2, 7, 6, 8, 9, 5, 4, 3|...], [3, 4, 9, 1, 2, 7, 5|...], [8, 5, 1, 3, 4, 6|...], [7, 6, 2,
 4, 8|...], [9, 1, 5, 2|...], [4, 3, 8|...], [6, 9|...], [1|...], [...|...]],
A = [2, 7, 6, 8, 9, 5, 4, 3, 1],
B = [3, 4, 9, 1, 2, 7, 5, 6, 8],
C = [8, 5, 1, 3, 4, 6, 2, 7, 9],
D = [7, 6, 2, 4, 8, 9, 1, 5, 3],
E = [9, 1, 5, 2, 7, 3, 6, 8, 4],
F = [4, 3, 8, 6, 5, 1, 7, 9, 2],
G = [6, 9, 4, 7, 3, 2, 8, 1, 5],
H = [1, 2, 3, 5, 6, 8, 9, 4, 7],
I = [5, 8, 7, 9, 1, 4, 3, 2, 6].

?-
```

Sample Input

```
Puzzle = [
    [2,_,_,_,9,_,_,_,1],
    [3,_,9,_,_,7,_,_,_],
    [_,_,1,_,4,_,_,7,_],
    [_,6,_,_,_,_,_,_,_],
    [_,_,_,_,_,3,_,_,_],
    [_,_,8,6,_,_,7,9,_],
    [6,_,_,7,_,_,8,_,_],
    [1,2,3,_,_,8,_,_,_],
    [_,8,7,_,_,4,3,_,_]],
Puzzle = [A,B,C,D,E,F,G,H,I],
sudoku(Puzzle).
```

Sample Output

```
A = [2, 7, 6, 8, 9, 5, 4, 3, 1],
B = [3, 4, 9, 1, 2, 7, 5, 6, 8],
C = [8, 5, 1, 3, 4, 6, 2, 7, 9],
D = [7, 6, 2, 4, 8, 9, 1, 5, 3],
E = [9, 1, 5, 2, 7, 3, 6, 8, 4],
F = [4, 3, 8, 6, 5, 1, 7, 9, 2],
G = [6, 9, 4, 7, 3, 2, 8, 1, 5],
H = [1, 2, 3, 5, 6, 8, 9, 4, 7],
I = [5, 8, 7, 9, 1, 4, 3, 2, 6].
```

Conclusion

It is seen that the sudoku puzzle problem can be easily solved by making it a constraint satisfaction problem. Moreover, SWI-Prolog's **clpfd** library allows us to set the constraints and get the solution at a very easy and fast way.

Finally, the source code is just around 25 lines, which is also easy to implement as well as easy to understand.