106119029

Lab 6 AI/ML

Propositional Logic (PL) is a branch of mathematical logic used to reason the truth and falsehood of logical expressions. (Ullman, Chapter 12: Propositional Logic, 2015) A propositional formula is made up of other propositions and the truth value of the formula is defined by the truth value of the propositions that it is made up of. Collab Link

Class for Suduko

The variable K(r, c, n) was selected for this purpose. In the variable, r stands for row, c for column and n for the number.

The row constraint, column constraint, and sub-grid constraints make sure that every row, column and sub-grid of the Sudoku puzzle should consist of every number from 1 to 9.

The number constraint establishes that every cell on the board should contain some number from 1 to 9.

And the duplicate constraint verifies that in the solution, no cell contains more than one number.

We will make a class with the following functions

1. **Puzzle** - Prints grid of the sudoku.
2. **Row-Check** - Checks the row constraint.
3. **Column-Check** - Checks the column constraint.
4. **Sub-Grid-Check** - Checks the sub grid constraint.
5. **Number-Constraint-Check** - Checks if all numbers follow the number constraint.
6. **Check-All-Constraint** - Checks if all the constraints are true.

Constraints summary.

Constraint	Formula
Number Constraint	$\bigwedge_{r=1}^{9}\left(\bigwedge_{c=1}^{9}\left(\bigvee_{n=1}^{9} K(r,c,n)\right)\right)$
Row Constraint	$\bigwedge_{r=1}^{9}\left(\bigwedge_{n=1}^{9}\left(\bigvee_{c=1}^{9} K(r,c,n)\right)\right)$
Column Constraint	$\bigwedge_{c=1}^{9}\left(\bigwedge_{n=1}^{9}\left(\bigvee_{r=1}^{9} K(r,c,n)\right)\right)$
Duplicate Constraint	$\bigwedge_{r=1}^{9}\left(\bigwedge_{c=1}^{9}\left(\bigvee_{n=1}^{9}\left(\bigwedge_{n'=n+1}^{9} K(r,c,n) \rightarrow \neg K(r,c,n')\right)\right)\right)$
Top left sub-grid Constraint	$\bigwedge_{n=1}^{9}\left(\bigvee_{r=1}^{9}\left(\bigvee_{c=1}^{9} K(r,c,n)\right)\right)$

Code:

```
class Sudoku:
 def __init__(self, dimension):
     self.M = dimension

 def puzzle(self, grid):
     for i in range(self.M):
         for j in range(self.M):
             print(grid[i][j], end=" ")
             if (j-2)%3 == 0:
               print(" ",end=" ")
         if (i-2)%3 == 0:
           print()
         print()

 def row_constraint_check(self, grid, row, num):
```

```
    for x in range(9):
        if(grid[row][x] == num):
            return False
    return True

def col_constraint_check(self, grid, col, num):
    for x in range(9):
        if(grid[x][col] == num):
            return False
    return True

def sub_grid_constraint_check(self, grid, row, col,  num):
    startRow = row - row % 3
    startCol = col - col % 3
    for i in range(3):
        for j in range(3):
            if grid[i + startRow][j + startCol] == num:
                return False
    return True

def check_all_constraint(self, grid, row, col, num):
    return self.row_constraint_check(grid, row, num) \
        and self.col_constraint_check(grid, col, num) \
        and self.sub_grid_constraint_check(grid, row, col, num)

def number_constraint_check(self, grid):
        for r in range(self.M):
            for c in range(self.M):
                if(not (grid[r][c] >= 1 and grid[r][c] <= 9)):
                    return False

def sudoku(self, grid, row, col):
    M = self.M
    if (row == M - 1 and col == M):
        return True
    if col == M:
        row += 1
        col = 0
    if grid[row][col] > 0:
        return self.sudoku(grid, row, col + 1)
```

```
        for num in range(1, M + 1, 1):

            if self.check_all_constraint(grid, row, col, num):
                grid[row][col] = num
                if self.sudoku(grid, row, col + 1):
                    return True
            grid[row][col] = 0
        return False

'''0 means the cells where no value is assigned'''
grid = [
    [0, 2, 0, 5, 0, 1, 0, 9, 0],
    [8, 0, 0, 2, 0, 3, 0, 0, 6],
    [0, 3, 0, 0, 6, 0, 0, 7, 0],
    [0, 0, 1, 0, 0, 0, 6, 0, 0],
    [5, 4, 0, 0, 0, 0, 0, 1, 9],
    [0, 0, 2, 0, 0, 0, 7, 0, 0],
    [0, 9, 0, 0, 3, 0, 0, 8, 0],
    [2, 0, 0, 8, 0, 4, 0, 0, 7],
    [0, 1, 0, 9, 0, 7, 0, 6, 0]
]
s = Sudoku(9)

print("For the following grid\n")
for i in range(9):
 for j in range(9):
    print(grid[i][j], end=" ")
    if (j-2)%3 == 0:
      print(" ",end=" ")
 if (i-2)%3 == 0:
    print()
 print()


if (s.sudoku(grid, 0, 0)):
    print("\nThe Following is the solution for the grid\n")
    s.puzzle(grid)
else:
    print("Solution does not exist:(")
```

Output:

```
For the following grid

0 2 0   5 0 1   0 9 0
8 0 0   2 0 3   0 0 6
0 3 0   0 6 0   0 7 0

0 0 1   0 0 0   6 0 0
5 4 0   0 0 0   0 1 9
0 0 2   0 0 0   7 0 0

0 9 0   0 3 0   0 8 0
2 0 0   8 0 4   0 0 7
0 1 0   9 0 7   0 6 0


The Following is the solution for the grid

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

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

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