Experiment-8
Implementation of knowledge representation schemes - use cases

Tamojit Sarkar
RA1811027010034
CSE-BD Sec-I2

Aim: To implement knowledge representation schemes- use case (Sudoku)

Code:

```
SIZE = 9
#sudoku problem
#cells with value 0 are vacant cells
matrix = [
    [6,5,0,8,7,3,0,9,0],
    [0,0,3,2,5,0,0,0,8],
    [9,8,0,1,0,4,3,5,7],
    [1,0,5,0,0,0,0,0,0],
    [4,0,0,0,0,0,0,0,2],
    [0,0,0,0,0,0,5,0,3],
    [5,7,8,3,0,1,0,2,6],
    [2,0,0,0,4,8,9,0,0],
    [0,9,0,6,2,5,0,8,1]]

#function to print sudoku
def print_sudoku():
    for i in matrix:
        print (i)

#function to check if all cells are assigned or not
#if there is any unassigned cell
#then this function will change the values of
#row and col accordingly
def number_unassigned(row, col):
    num_unassign = 0
    for i in range(0,SIZE):
        for j in range (0,SIZE):
            #cell is unassigned
            if matrix[i][j] == 0:
                row = i
                col = j
                num_unassign = 1
                a = [row, col, num_unassign]
                return a
    a = [-1, -1, num_unassign]
    return a
```

```
#function to check if we can put a
#value in a paticular cell or not
def is_safe(n, r, c):
    #checking in row
    for i in range(0,SIZE):
        #there is a cell with same value
        if matrix[r][i] == n:
            return False
    #checking in column
    for i in range(0,SIZE):
        #there is a cell with same value
        if matrix[i][c] == n:
            return False
    row_start = (r//3)*3
    col_start = (c//3)*3;
    #checking submatrix
    for i in range(row_start,row_start+3):
        for j in range(col_start,col_start+3):
            if matrix[i][j]==n:
                return False
    return True

#function to check if we can put a
#value in a paticular cell or not
def solve_sudoku():
    row = 0
    col = 0
    #if all cells are assigned then the sudoku is already solved
    #pass by reference because number_unassigned will change the values of row and col
    a = number_unassigned(row, col)
    if a[2] == 0:
        return True
    row = a[0]
    col = a[1]
    #number between 1 to 9
    for i in range(1,10):
        #if we can assign i to the cell or not
        #the cell is matrix[row][col]
        if is_safe(i, row, col):
            matrix[row][col] = i
            #backtracking
            if solve_sudoku():
                return True
            #f we can't proceed with this solution
            #reassign the cell
            matrix[row][col]=0
    return False

if solve_sudoku():
    print_sudoku()
else:
    print("No solution")
```

Output:

```
[6, 5, 1, 8, 7, 3, 2, 9, 4]
[7, 4, 3, 2, 5, 9, 1, 6, 8]
[9, 8, 2, 1, 6, 4, 3, 5, 7]
[1, 2, 5, 4, 3, 6, 8, 7, 9]
[4, 3, 9, 5, 8, 7, 6, 1, 2]
[8, 6, 7, 9, 1, 2, 5, 4, 3]
[5, 7, 8, 3, 9, 1, 4, 2, 6]
[2, 1, 6, 7, 4, 8, 9, 3, 5]
[3, 9, 4, 6, 2, 5, 7, 8, 1]
```

Result: We have successfully implemented a knowledge representation scheme in the Sudoku use case.