INDEX

5th Sem "B"

Name Ibrahim Iftekhar Khan (Batch 1) Sec

Roll No. Subject School/College

School/College Tel. No. Parents Tel. No.

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
1	3/11/23	Python program on Kaggle		
2	10/11/23	Practice program		
3	17/11/23	Prog. for Tic-Tac-Toe game		
4	24/11/23	Prog. for puzzle game		
5	8/12/23	8 puzzle prog. using iterative		
~~4~~		deepening search		
6		8 puzzle prog. A*		
~~7~~		8 puzzle prog BFS		
8	22/12/23	Vaccum cleaner		
9	29/12/23	KB using propositional logic		
10	19/01/23	unification		
11	19/01/23	program to convert FOL to CNF		
12	19/01/23	prog. for forward chaining		

completed!

Date ___/___/___
Page _____

Q 1 Write a program for Tic-Tac-Toe game:

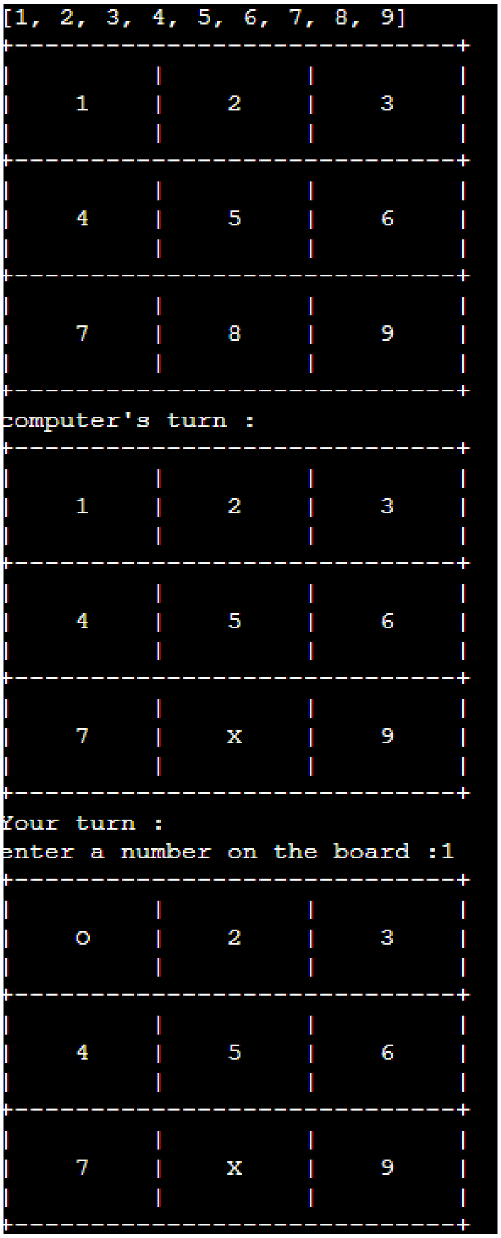
```
import random
tic=[1,2,3,4,5,6,7,8,9]

def printBoard(tic):
    print(tic[0]+'|'+tic[1]+'|'+tic[2])
    print("-----")
    print(tic[3]+'|'+tic[4]+'|'+tic[5])
    print("-----")
    print(tic[6]+'|'+tic[7]+'|'+tic[8])

def iswinner(tic,pos):
    if tic[0]==tic[4] and tic[4]==tic[8] or
        tic[2]==tic[4] and tic[4]==tic[6]
        return True
    else if tic[pos-0]==tic[pos-3] and tic[pos-3]==
        return True                 tic[pos-6]
    else if tic[pos//3+1]==tic[pos//3+2] and tic[pos//3+2]
                                        ==tic[pos//
                                        3+3]

    return True

return False
def update_user(tic):
    num = int(input("Enter a number on the board"))
    while(num not in tic):
        num = int(input("Enter a number on the board
    tic[num-1]=='0'

def update_comp(tic):
    for i in tic:
        if i!='x' and i!='0':
            tic[i-1]='x'
            if(iswin(tic,i-1)==True):
```

else
tic[i-1] = i

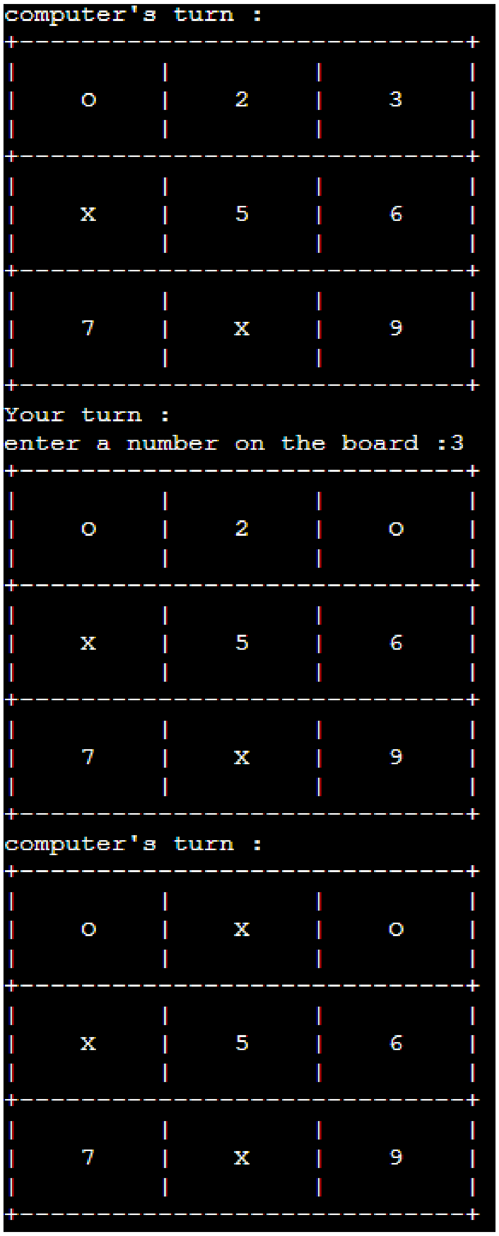
algorithm :-

(i) make a board ad initialize the value
(ii) Make a winner function check for the winner possibilities
(iii) Make a user function which will take no as input.
(iv) make a computer function.
  (i) Maximize computer wins
  (ii) Maximize user wins
  (iii) generate random position.
(v) Now check for any combination after 5 steps as there can be a 3 combination
(vi) put main function
  put the board
  ad shift b/w the user & computer.

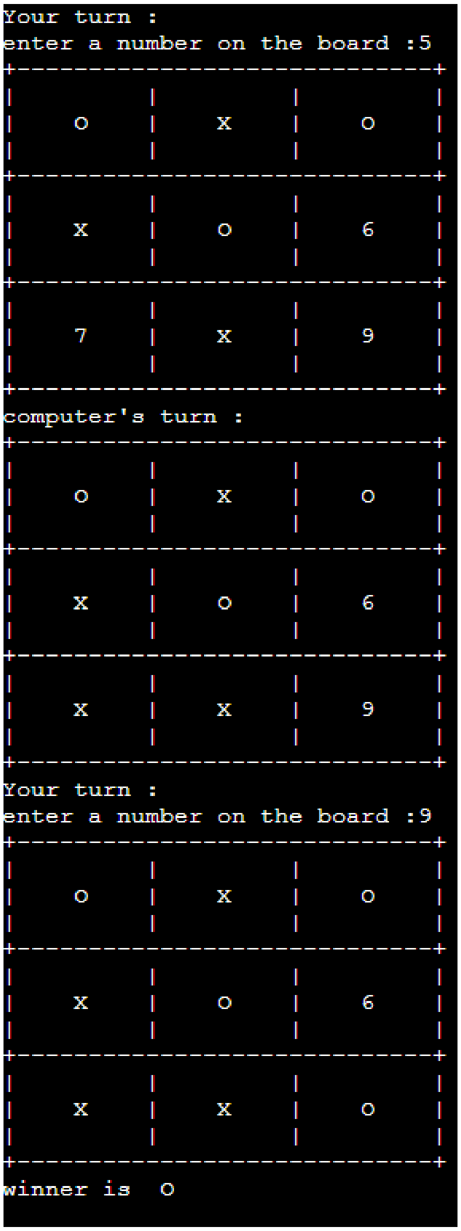
output:-



```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
+-----------------------------+
|         |         |         |
|    1    |    2    |    3    |
|         |         |         |
+-----------------------------+
|         |         |         |
|    4    |    5    |    6    |
|         |         |         |
+-----------------------------+
|         |         |         |
|    7    |    8    |    9    |
|         |         |         |
+-----------------------------+
computer's turn :
+-----------------------------+
|         |         |         |
|    1    |    2    |    3    |
|         |         |         |
+-----------------------------+
|         |         |         |
|    4    |    5    |    6    |
|         |         |         |
+-----------------------------+
|         |         |         |
|    7    |    X    |    9    |
|         |         |         |
+-----------------------------+
Your turn :
enter a number on the board :1
+-----------------------------+
|         |         |         |
|    O    |    2    |    3    |
|         |         |         |
+-----------------------------+
|         |         |         |
|    4    |    5    |    6    |
|         |         |         |
+-----------------------------+
|         |         |         |
|    7    |    X    |    9    |
|         |         |         |
+-----------------------------+
```



```
computer's turn :
+-----------------------------+
|         |         |         |
|    O    |    2    |    3    |
|         |         |         |
+-----------------------------+
|         |         |         |
|    X    |    5    |    6    |
|         |         |         |
+-----------------------------+
|         |         |         |
|    7    |    X    |    9    |
|         |         |         |
+-----------------------------+
Your turn :
enter a number on the board :3
+-----------------------------+
|         |         |         |
|    O    |    2    |    O    |
|         |         |         |
+-----------------------------+
|         |         |         |
|    X    |    5    |    6    |
|         |         |         |
+-----------------------------+
|         |         |         |
|    7    |    X    |    9    |
|         |         |         |
+-----------------------------+
computer's turn :
+-----------------------------+
|         |         |         |
|    O    |    X    |    O    |
|         |         |         |
+-----------------------------+
|         |         |         |
|    X    |    5    |    6    |
|         |         |         |
+-----------------------------+
|         |         |         |
|    7    |    X    |    9    |
|         |         |         |
+-----------------------------+
```



```
Your turn :
enter a number on the board :5
+-----------------------------+
|         |         |         |
|    O    |    X    |    O    |
|         |         |         |
+-----------------------------+
|         |         |         |
|    X    |    O    |    6    |
|         |         |         |
+-----------------------------+
|         |         |         |
|    7    |    X    |    9    |
|         |         |         |
+-----------------------------+
computer's turn :
+-----------------------------+
|         |         |         |
|    O    |    X    |    O    |
|         |         |         |
+-----------------------------+
|         |         |         |
|    X    |    O    |    6    |
|         |         |         |
+-----------------------------+
|         |         |         |
|    X    |    X    |    9    |
|         |         |         |
+-----------------------------+
Your turn :
enter a number on the board :9
+-----------------------------+
|         |         |         |
|    O    |    X    |    O    |
|         |         |         |
+-----------------------------+
|         |         |         |
|    X    |    O    |    6    |
|         |         |         |
+-----------------------------+
|         |         |         |
|    X    |    X    |    O    |
|         |         |         |
+-----------------------------+
winner is  O
```

8 puzzle bfs
Code:

8 Program to Solve the 8-puzzle using BFS.

```
from collections import deque   ) double ended queue.

def find_blank(board,:          to find zero (blank) space
    for i in range(3):
        for j in range(3):
            if board[i][j] == 0:
                return i, j

def generate_moves(board):      function to generate possible moves for a given state
    moves = []
    blank_row, blank_col = find_blank(board)
    possible_moves = [
        (1,0), (-1,0), (0,1), (0,-1)
        up     down    right  left
    ]
    for dr, dc in possible_moves:
        new_row, new_col = blank_row+dr, blank_col+d
        if 0<=new_row<3 and 0<=new_col<3:      check if new position is within the board bounds
            new_board = [row[:] for row in board]
            new_board[blank_row][blank_col],
            new_board[new_row][new_col] = new_board[new_row][new_col], new_board[blank_row][blank_col]   copy with update
            moves.append(new_board)
    return moves.

BFS
def solve_puzzle(initial_state, goal_state):
    visited = set()
    queue = deque([(initial_state, []))]   state & their paths
```

```
    while queue:
        current_state, path = queue.popleft()
        visited.add(tuple(map(tuple, current_state)))

        if current_state == goal_state:
            return path

        possible_moves = generate_moves(current_state)
        for move in possible_moves:
            if tuple(map(tuple, move)) not in visited:
                queue.append((move, path+[move]))

    return None

def print_steps(solution_path):
    if solution_path:
        print("Steps to reach the goal:")
        for step in solution_path:
            print("-----")
            for row in step:
                print("|", end=" ")
                for val in row:
                    if val == 0:
                        print(" ", end=" | ")
                    else:
                        print(val, end=" | ")
                print()
            print("-----")
            print()
    else:
        print("No solution")
```

initial = [
[1, 2, 3], [4, 0, 5], [6, 7, 8]]

goal = [[0, 1, 2], [3, 4, 5], [6, 7, 8]]

solution_path = solve_puzzle(initial, goal)
print_steps (solution_path)

1	2	3
0	4	5
6	7	8

① will use using a doubleend queue

② find the blankspace here (0) using function
for i in range 3:
for j in range3
if board[i][j]==0 :
return i, j

③ Now generate moves using function it move the blankspace in all possible ways ↑↓←→
(i) first check if new position is in board bound (3x3)

(ii) if it is in bound generate the move ie move blank space with adjacent tiles and board for path.

Bfs/ Use bfs now use visted list
every time generate a copy and check with goalstate.

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

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

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

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

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

Write proper

24/11

output:-

main.py

```
    d=[]
    if b not in [0,1,2]:
        d.append('u')
    if b not in [6,7,8]:
        d.append('d')
    if b not in [0,3,6]:
        d.append('l')
    if b not in [2,5,8]:
        d.append('r')

```

```
1 | 2 | 3
4 | 5 | 6
0 | 7 | 8
-----------
1 | 2 | 3
0 | 5 | 6
4 | 7 | 8
-----------
1 | 2 | 3
4 | 5 | 6
7 | 0 | 8
-----------
0 | 2 | 3
1 | 5 | 6
4 | 7 | 8
-----------
1 | 2 | 3
5 | 0 | 6
4 | 7 | 8
-----------
1 | 2 | 3
4 | 0 | 6
7 | 5 | 8
-----------
1 | 2 | 3
4 | 5 | 6
7 | 8 | 0
-----------
Success


...Program finished with exit code 0
Press ENTER to exit console.
```

8 puzzle iterative deepening search:-
Code:-

Date __/__/__
Page _____

Lab - 5

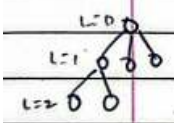
8/12/2023

3 8 puzzle problem using iterative deepening search algorithm.

(1) Node(data, level) : initialize node with puzzle state & level.

(2) def puzzle:
→ output () ← Start & goal state
→ dls () (node, goal, depth) ← perform dls

(3) def dls (node, goal, depth)
if (current state == goal):
return solution



else:
generate child node, recursively call with increase level

(4) def IDS (Start, goal)
→ Start with depth : 0
→ repeat until goal is found:
→ perform DLS with current depth
→ if Sol^n found ← exit
increment depth.

(5) → generate puzzle instance
→ call IDS (Start, goal)

```
def id_dfs (puzzle, goal, get-moves):
    import itertools
    def dfs (route, depth):
        if depth ==0 :
            return
        if route [-1] == goal:
            return route
        for move in get_moves (route[-1]):
            if move not in route:
                next_route = dfs (route[from], dep
                if next_route:
                    return next_route
    for depth in itertools.count():
        route = dfs([puzzle], depth)
        if route:
            return route

def possible_moves (state):
    b = state.index (0)
    d = []
    if b not in [0, 1, 2]:
        d.append ('u')
    if b not in [6,7,8]:
        d.append ('d')
    if b not in [0,3,6]:
        d.append (l)
    pos_moves = []
    for i in d:
        pos_moves.append (generate(state, i, b))
    return pos_moves.

initial = [1, 2, 3, 0, 4, 6, 7, 5, 8]
goal = [1, 2, 3, 4, 5, 6, 7, 8, 0]
route = id_dfs (initial, goal, possible_moves)
if route:
    print (" Success")
    print ("Path: ", path)
else:
    print ("Failed to find Solution")
```

Output:-
Success !!
path:
[[1,2,3,0,4,6,7,5,8],
[1,2,3,4,0,6,7,5,8],
[1,2,3,4,5,6,7,0,8],
[1,2,3,4,5,6,7,8,0]]

output:-

main.py

```python
        d.append('l')
    if b not in [2, 5, 8]:
        d.append('r')

    pos_moves = []
    for i in d:
        pos_moves.append(generate(state, i, b))
    return pos_moves


def generate(state, m, b):
    temp = state.copy()

    if m == 'd':
        temp[b + 3], temp[b] = temp[b], temp[b + 3]
    if m == 'u':
        temp[b - 3], temp[b] = temp[b], temp[b - 3]
    if m == 'l':
        temp[b - 1], temp[b] = temp[b], temp[b - 1]
    if m == 'r':
        temp[b + 1], temp[b] = temp[b], temp[b + 1]

    return temp


# calling ID-DFS
initial = [1, 2, 3, 0, 4, 6, 7, 5, 8]
goal = [1, 2, 3, 4, 5, 6, 7, 8, 0]

route = id_dfs(initial, goal, possible_moves)

if route:
    print("Success!! It is possible to solve 8 Puzzle problem")
    print("Path:", route)
else:
```

input

```
Success!! It is possible to solve 8 Puzzle problem
Path: [[1, 2, 3, 0, 4, 6, 7, 5, 8], [1, 2, 3, 4, 0, 6, 7, 5, 8], [1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 7, 8, 0]]


...Program finished with exit code 0
Press ENTER to exit console.
```

8 puzzle A*
Code:-

open closed

Date __/__/__
Page _____

4 8 puzzle problem using A* algorithm:

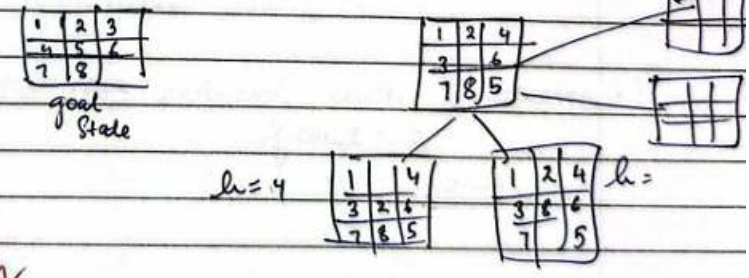
(1) Node (data, level) : initialize node with puzzle state & level

(2) def puzzle():
- accept(): accept start state & goal state
- f(start, goal) ← calculate f = h+g
- process() ← process A* algorithm

(3) · Heuristic

h(start, goal) : No of Misplaced tiles.

(4) A* Algorithm:
1) initialize start & goal state & add to open list
2) loop until goal state comes
→ select node have lowest from open list
→ Display state & check goal
→ generate child nodes
→ add current node which is done with expanding to closed list. add the ones we dont want to continue.
→ Explore list
→ Display moves.



```
class Node:
    def __init__(self, data, level, fval):
        self.data = data
        self.level = level
        self.fval = fval

    def generate_child(self):
        x,y = self.find(self.data, '_')
        val_list = [[x, y-1], [x, y+1], [x-1, y], [x+1, y]]
        children = []
        for i in val_list:
            child = self.shuffle(self.data, x, y, i[0], i[1])
            if child is not None:
                child_node = Node(child, self.level+1,
                children.append(child_node)
        return children

    def shuffle(self, puz, x1, y1, x2, y2):
        if x
```

output:-

main.py

```
        print("Enter the goal state matrix \n")
        goal = self.accept()

        start = Node(start,0,0)
        start.fval = self.f(start,goal)
        """ Put the start node in the open list"""
```

```
Enter the start state matrix

1 2 3
4 5 6
_ 7 8
Enter the goal state matrix

1 2 3
4 5 6
7 8 _




  |
  |
 \'/

1 2 3
4 5 6
_ 7 8
_

  |
  |
 \'/

1 2 3
4 5 6
7 _ 8

  |
  |
 \'/

1 2 3
4 5 6
7 8 _


...Program finished with exit code 0
Press ENTER to exit console.
```

Vacuum cleaner:-
Code:-

Vacuum Cleaner Problem.

Date __/__/__
Page _____

5

algorithm:-

define room A & B

① initial State → Either in room A or B

② Actions → move L, move R, Suck (clean)

Succesor function :- where the machine reaches after the action is performed.

① Result (S, a)
State action

② ~~After~~ first it checks in the room its in whether its clean or not if clean it moves to next (with move function).

③ it runs the goal state function after every step or action.

if room A == clean && room B == clean
print ("cleaning complete")

④ if not clean it runs suck function and returns clean then moves to the next.

⑤ We have path cost counter. (each step 1)

if count == 2
end.

Code:-

```
class VaccumCleaner:
    def __init__(self):
        self.location = random.choice(['A', 'B'])

    def move_left(self):
        print("Moving left")
        self.location = 'A'

    def move_right(self):
        print("Moving right")
        self.location = 'B'

    def suck(self, room):
        print(f"Sucking dirt in Room{room}")
        return 'clean'

def simulate_cleaning():
    vacuum = VaccumCleaner()
    rooms = {
        'A': random.choice(['clean', 'dirty']),
        'B': random.choice(['clean', 'dirty'])
    }

    print("initial State:")
    print(f"Vacuum cleaner is in Room{vacuum.location}")
    print(f"Room A: {rooms['A']}")
    print(f"Room B: {rooms['B']}\n")
```

```
if room['A'] == 'clean' and rooms['B'] == 'clean':
        print ("Both rooms are clean, No cleaning
                                        needed)
else
        print ("Starting the cleaning process ...")
        current_room = vacuum.location
        cleaned_room = vacuum.suck(current_room)

        if cleaned_room == 'clean':
                rooms[current_room] = 'Clean

        if current_room == 'A'
                vacuum.move_right()
                current_room = 'B'
        else:
                vacuum.move_left()
                current_room = 'A'

        print ("\n Cleaning completed.")
        print (f" Final State ")
        print (f" Room A : {rooms['A']}")
        print (f" Room B: {rooms['B']}")

Simulate_cleaning()
```

Output :-

```
Enter initial location of vacuum cleaner (A/B): A
Enter State for Room A (clean/dirty): dirty
Enter State for Room B (clean/dirty): dirty
Initial State:-
Vacuum Cleaner is in Room A
Room A: dirty
Room B: dirty
```

Starting the cleaning process

Sucking dirt in Room A

moving right

Sucking dirt in Room B

Cleaning completed.

Final State:

Vacuum cleaner is in Room B

Room A : Clean

Room B : Clean

Output:-

main.py

```python
        nonlocal cost
        if goal_state[room] == 1:
            print(f"Cleaning Room {room}...")
            goal_state[room] = 0
            cost += 1  # Cost for cleaning
            print(f"Room {room} has been cleaned. Current cost: {cost}")
        else:
            print(f"Room {room} is already clean.")

    # Cleaning logic
    rooms = ['A', 'B', 'C', 'D']
    current_index = rooms.index(location_input)

    # Clean all rooms starting from the initial location
    for i in range(current_index, len(rooms)):
        clean_room(rooms[i])

    # Clean remaining rooms (if the initial location was not 'A')
    for i in range(0, current_index):
        clean_room(rooms[i])

```

input

```
Enter Initial Location of Vacuum (A/B/C/D): B
Enter status of each room (1 - dirty, 0 - clean):
Status of Room A: 1
Status of Room B: 0
Status of Room C: 1
Status of Room D: 1
Initial Location Condition: {'A': 1, 'B': 0, 'C': 1, 'D': 1}
Room B is already clean.
Cleaning Room C...
Room C has been cleaned. Current cost: 1
Cleaning Room D...
Room D has been cleaned. Current cost: 2
Cleaning Room A...
Room A has been cleaned. Current cost: 3
Final State of Rooms: {'A': 0, 'B': 0, 'C': 0, 'D': 0}
Performance Measurement (Total Cost): 7


...Program finished with exit code 0
Press ENTER to exit console.
```

Knowledge-based entailment:-
Code:-

Q 6 Entailment

Inputs:-

Knowledge base (set of logical rules)
Query statement

Steps:-

1.) Negate the query:
Obtain the negation.

2.) Combine with knowledge base:

3.) Check satisfiability:
to check if the negation with KB is satisfying the rules

4.) Determine entailment
if conjunction is not satisfiable → True
if conjunction is satisfiable → False.

Proved

code:-

```
from sympy import symbols
def create knowledge base()
    p = symbols ('p')
    q = symbols ('q')
    r = symbols ('r')
    knowledge base = And (implies(p,q), implies(q,r),
                                        Not (r))

    return knowledge base.

def query - entails (knowledgebase, query)
    entailment = satisfiable (And ( knowledgebase,
                                        Not (query))

    return not entailment.

if - name_ = '__main__'
    kb = create knowledge base ()
    query = symbols ('p')
    result = query - entails (kb, query)
    print (" knowledge Base ", kb)
    print ("Query", query)
    print (" Query entailsknowledge base", result)
```

Output:-

Knowledgebase : ~r & (Implies (p, q) & (Implies(q,r

Query : p

Query entails knowledgebase, False.

output:-

main.py

```python
    entailment = satisfiable(And(knowledge_base, Not(query)))

    # If there is no satisfying assignment, then the query is entailed
    return not entailment

if __name__ == "__main__":
    # Create the knowledge base
    kb = create_knowledge_base()

    # Define a query
    query = symbols('p')

    # Check if the query entails the knowledge base
    result = query_entails(kb, query)

    # Display the results
    print("Knowledge Base:", kb)
    print("Query:", query)
    print("Query entails Knowledge Base:", result)

```

```
Knowledge Base: ~r & (Implies(p, q)) & (Implies(q, r))
Query: p
Query entails Knowledge Base: False


...Program finished with exit code 0
Press ENTER to exit console.
```

Knowledge-based resolution
Code:-

7 Knowledge based resolution :-

Inputs :-

Knowledge base (set of clauses in propositional logic)

Steps :-

1.) Initialize resolvents :-
2.) Repeat until no new resolvents can be generated
2.1 pairwise selection
(clause A , clause B)

3.) Resolving clause.

$B \vee A$
$\neg C \vee A$
$\neg B \vee A$
$C \vee \neg D$
$\neg A \vee \neg B \vee D$
$\neg A \rightarrow$ Input

P
$\neg P \vee Q$
$P \vee \neg Q \vee R$
$\neg Q \vee R$
$P \vee R \vee \neg Q$

Negate the query & add it to the kb
→ Repeatedly resolve the pairs of clauses in the knowledge base until a contradiction found or No more rules are possible.

```
def negate-literal (literal):
    if literal [0] == 'n':
        return literal [1:]
    else:
        return 'n' + literal

def resolve ( C1, C2):
    resolve-clause = set(C1) | set(C2)

    for literal in C1:
        if negate-literal (literal) in C2:
            resolved-clause - remove (literal
            resolve clause - remove (negate
                literal (literal
            return tuple (resolved-clause)

def resolution (knowledge base):

    while true:
        new clauses = set()
        for i, C1 in enumerate (KB)
            if i != j
                new clause = resolve (C1, C2)
                if len (new clause) = 0 for
                    clause
                new-clause.add (new cla
            if not new-clause
                break

    Knowledgebas |= new-clause
    return knowledge-base

if - name = = " - main - ":
```

Page

```
kb = {('p', 'q'), ('~p', 'r'}, ('~q', '~r')}

result = resolution(kb)
print("original kb", kb)
print("Resolved kb", result)
```

29/12

Output:-

```
rules = 'PvQ ~PvR ~QvR' #P=vQ, P=>Q : ~PvQ, Q=>R, ~QvR
goal = 'R'
main(rules, goal)
```

```
Step    |Clause |Derivation
-----------------------------
 1.     | PvQ   | Given.
 2.     | ~PvR  | Given.
 3.     | ~QvR  | Given.
 4.     | ~R    | Negated conclusion.
 5.     | QvR   | Resolved from PvQ and ~PvR.
 6.     | PvR   | Resolved from PvQ and ~QvR.
 7.     | ~P    | Resolved from ~PvR and ~R.
 8.     | ~Q    | Resolved from ~QvR and ~R.
 9.     | Q     | Resolved from ~R and QvR.
 10.    | P     | Resolved from ~R and PvR.
 11.    | R     | Resolved from QvR and ~Q.
 12.    |       | Resolved R and ~R to Rv~R, which is in turn null.
A contradiction is found when ~R is assumed as true. Hence, R is true.
```

Unification
Code:-

8 Unification

Eg knows (John, x) Knows (John, Jane)
{x / Jane}

Step 1: If term 1 or term 2 is a variable or constant then:
a) term 1 or term 2 are identical
return NIL

b.) Else if term 1 is a variable
if term 1 occurs in term 2
return FAIL
c.) else if term 2 is a variable
if term 2 occurs in term 1
return FAIL
else
return {(term 1 / term 2)}
d.) else return FAIL

Step 2: if predicate (term 1) ≠ predicate (term 2)
return FAIL

Step 3: number of argument ≠
return FAIL

Step 4: set (SUB ST) to NIL

Step 5: For i = 1 to the number of elements in term 1
a) Call unify (ith term 1, ith term 2)
put results into S
S = FAIL
return FAIL.

Step 5: c.) if S ≠ NIL

a.) Apply S to the remainder of both L_1 & L_2

b.) SUBST = APPEND (S, SUBST)

Step 6: Return SUBST

① predicate same

② No of arguments

③ John John

X Jane

then put in subset

Prob

```
import re
def getInitialPredicate (expression):
    return expression.split("(")[0]
def isConstant (char):
    return char.isupper() and len(char)==1
def replaceAttributes (exp, old, new):
    attributes = getAttributes (exp)
    for index, val in enumerate (attributes).
        if val == old :
            attributes [index] = new
    predicate = getInitialPredicate (exp)
    return predicate + "(" + ",".join (attributes) + ")"
def apply (exp, substitutions):
    for substitution in substitutions:
        new, old = substitution
        exp = replaceAttributes (exp, old, new)
    return exp.

def getFirstPart (expression):
    predicate = getInitialPredicate (expression)
    attributes = getAttributes (expression)
    return newExpression.
```

```
defunify (exp1, exp2)
    if exp1 == exp2
        return false.
    if isConstant (exp1):
        return [(exp2, exp1)]
    if isVariable (exp1):
        if checkOccurs (exp1, exp2):
            return False
    attribute count1 = len (getAttributes (exp1))
    attribute Count2 = len (getAttributes (exp2))
    if attributeCount1 != attribute count2:
        return false.
    head1 = get firstPart (exp1)
    head2 = get firstPart (exp2)
    initial Substitution = unify (heads, heads2)
    if not initial Substitution:
        return False.
    return initial Substitution

    tail1 = getRemainPart (exp1)
    tail2 = getRemainingPart (exp2)

    initial Substitution . extend (remaining Substitution)
    return initial Substitution.
```

output:-

```
exp1 = "knows(X)"
exp2 = "knows(Richard)"
substitutions = unify(exp1, exp2)
print("Substitutions:")
print(substitutions)
```

```
Substitutions:
[('X', 'Richard')]
```

```
[7] exp1 = "knows(A,x)"
    exp2 = "knows(y,mother(y))"
    substitutions = unify(exp1, exp2)
    print("Substitutions:")
    print(substitutions)
```

```
Substitutions:
[('A', 'y'), ('mother(y)', 'x')]
```

Fol to cnf
Code:-

FOL to CNF

NOT A, OR

$\forall x \Rightarrow wet(x) \wedge snow(x) \rightarrow slip(x)$

FOL to CNF conversion

Date __/__/__
Page _____

Step1.) create a list of SKOLEM. CONSTANTS

Step2 Find $\forall, \exists \rightarrow$ there exists
↳ for every

if the attributes are lower case, replace them with a skolem constant.
remove used skolem constant or function from list
if the attributes are both lowercase and uppercase replace the appearance attributes with a Skolem function.

Step3 : replace $\Leftrightarrow$ with '–'
transform – as $P = (P \Rightarrow Q) \wedge (Q \Rightarrow P)$

Step4: replace $\Rightarrow$ with '–'

Step5 : Apply de morgan's law
replace ~[
: as ~P & ~Q if (, was present)
replace ~[
: as ~P | ~Q if (& was present)
replace ~ with " "

$\forall x$ King (x) $\wedge$ Greedy (x) $\Rightarrow$ Evil (x)
King (R) $\wedge$ Greedy (R) $\Rightarrow$ Evil (R)
~ [King (R) $\wedge$ greedy (R)] $\vee$ Evil (R)

~ [King (R)) $\vee$ ~ (Greedy (R)) $\vee$ Evil (R)

Proof goal

```
def getAttributes(string):
    expr = '([^)]+)'
    matches = re.findall(expr, string)
    return [m for m in str(matches) if m.isalpha()]

def getPredicates(string):
    expr = '[a-z~]+([A-Za-z]+)'
    return re.findall(expr, string)

def DeMorgan(sentence):
    string = ''.join(list(sentence).copy())
    string = string.replace('~~', '')
    flag = '[' in string
    string = string.replace('~[', '')
    for predicate in getPredicates(string):
        string = string.replace(predicate, f'~{predicate
    s = list(string)
    for i, c in enumerate(string):
        if c == '|':
            s[i] = '&'
        elif c == '&':
            s[i] = '|'
    string = ''.join(s)
    string = string.replace('~~', '')
    return f'[{string}]' if flag else string

def Skolemization(sentence):
    SKOLEM_CONSTANTS = [f'{chr(c)}' for c in range
                        (ord('A'), ord('Z')+1)]
    Statement = ''.join(list(sentence).copy())
    matches = re.findall('[∀∃].', Statement)
    for match in reversed(matches[::-1]):
```

```
for predicate in get predicates(state):
        attributes = get Attributes(predicate)
    if "".join(attributes).islower():
        Statement = Statement.replace(matches[1],
                                  Skolem_constants.pop(0))
    else:
        al = [a for a in attributes if a.islower()]
        au = [a for a in attributes if not a.islower()
        [0]
    return statement

while '-' in statement:
    i = statement.index('~∀')
    statement = list(statement)
    statement[i] = , statement[i+1], statement[i+2] = '∃
    statement[i+2], '~'
    statement = ''.join(statement)
statement = statement.replace('~[∀', '[~∀')
    exps = '(~[∀|∃])'
    for s in statements:
        exps = '~'
    for s in statements:
        statement = statement.replace(s, exps
    return statement.
```

output:-

```
print(Skolemization(fol_to_cnf("animal(y)<=>loves(x,y)")))
print(Skolemization(fol_to_cnf("∀x[∀y[animal(y)=>loves(x,y)]]=>[∃z[loves(z,x)]]")))
print(fol_to_cnf("[american(x)&weapon(y)&sells(x,y,z)&hostile(z)]=>criminal(x)"))
```

```
[~animal(y)|loves(x,y)]&[~loves(x,y)|animal(y)]
[animal(G(x))&~loves(x,G(x))]|[loves(F(x),x)]
[~american(x)|~weapon(y)|~sells(x,y,z)|~hostile(z)]|criminal(x)
```

Forward chaining
Code:-

19/1/2024

1° Forward chaining

1.) Input the knowledge base and the query
2.) for i in KB:
    if i == query return True
        if l ⇒ r
            Split lhs and rhs part
            if lhs in KB:
                add rhs to KB
    return False
3.) To remove variables
        if i.lower():
            replace the variable with constants

Example:
KB
King (x) & greedy (x) ⇒ evil (x)
King (John)
greedy (John)
King (Richard)

Query
evil(x)

Code:-

```
import re

def is variable(x):
    return len(x)==1 and x.islower() and x.isalpha()

def getAttributes(string):
    expr = '\([^)]+\)'
    matches = re.findall(expr, string)
    return matches

def get predicates(string):
    expr = '([a-zA~]+)\([^&|]+\)'
    return re.findall(expr, string)

class Fact:
    def __init__(self, expression):
        self.expression = expression
        self.params = params
        self.result = any(self.getconstants())

    def splitExpression(self, expression):
        predicate = get predicates(expression)[0]
        params = getAttributes(expression[0]
                 strip('()').split(',')
        return [predicate, params]

    def getresult(self):
        return self.result

    def getConstants(self):
        return [None if isvariable(c) else c for
        c in self.params]
```

```
def get variable(self):
        return [v if isVariable(v) else None for v in
                self.parms]

Class Implication:
    def __init__(self, expression):
        self.expression = expression
        l = expression.split('=>')
        self.lhs = [fact(f) for f in l[0].split]
        self.rhs = fact(l[1])

    def evaluate(self, facts):
        constants = {}
        new_lhs = []
        for fact in facts:
            for val in self.lhs
                if val.predicate == fact.predicate
                    for i,v in enumerate(val.getv
                                        ariables
                                        ()):

Class KB:
    def __init__(self):
        self.facts = set()
        self.implications = set()

    def tell(self, c):
        if => in c:
            self.implications.add(Implication

        for i in self.implications:
            res = i.evaluate(self.facts)
```

```
def query (self, o):
    facts = set ([f.expression for in self.facts])

def display (self):
    print ("All facts")
        for i, in enumerate (set ([f.expression
                for in self.facts
            print (f"{i+1} {s}")

kb = KB()

KB - tell ("King(x) & greedy(x) => evil(x))
kb = tell ('King(John)')
kb - tell ('greedy(John)')
kb - tell ('King(Richard)')
kb - query ('evil(x)')
```

Output:

Querying evil(x):

evil (John)

29/1/24

Complete!

Output:-

```
kb_ = KB()
kb_.tell('king(x)&greedy(x)=>evil(x)')
kb_.tell('king(John)')
kb_.tell('greedy(John)')
kb_.tell('king(Richard)')
kb_.query('evil(x)')
```

```
Querying evil(x):
        1. evil(John)
```