

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT

on

ARTIFICIAL INTELLIGENCE

Submitted by

NISARGA GONDI (1BM21CS069)

Under the Guidance of

Prof. Swathi Sridharan
Assistant Professor, BMSCE

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING



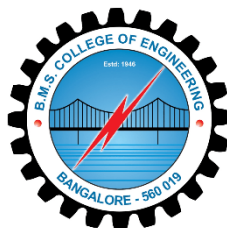
B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

November 2023-February 2024

**B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering**



CERTIFICATE

This is to certify that the Lab work entitled “**Artificial Intelligence**” carried out by **Nisarga Gondi (1BM21CS069)** , who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023-24.

The Lab report has been approved as it satisfies the academic requirements in respect of **Artificial Intelligence- (22CS5PCAIN)** work prescribed for the said degree.

Prof. Swathi Sridharan
Assistant professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

B. M. S. COLLEGE OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING



DECLARATION

I, Nisarga Gondi (1BM21CS069), student of 5th Semester, B.E, Department of Computer Science and Engineering, B. M. S. College of Engineering, Bangalore, here by declare that, this lab report entitled " **Artificial Intelligence**" has been carried out by me under the guidance of **Prof. Swathi Sridharan**, Assistant Professor, Department of CSE, B. M. S. College of Engineering, Bangalore during the academic semester November-2023-February-2024.

I also declare that to the best of my knowledge and belief, the development reported here is not from part of any other report by any other students.

TABLE OF CONTENTS

Sl. No.	Title	Page No.
1	Implementation of Tic Tac Toe	– 5-9
2	8 Puzzle Breadth First Search Algorithm	10-12
3	8 Puzzle Iterative Deepening Search Algorithm	13-15
4	8 Puzzle A* Search Algorithm	16-20
5	Vacuum Cleaner	21-22
6	Knowledge Base Entailment	23-24
7	Knowledge Base Resolution	25-28
8	Unification	29-33
9	FOL to CNF	34-35
10	Forward reasoning	36-39

Program 1

Implementation of tic tac toe

Code

```
# Create a 3x3 tic tac toe board of "" strings for each value
board = [' '] * 9

# Create a function to display your board
def display_board(board):
    print(f" {board[0]} | {board[1]} | {board[2]} ")
    print("---+---+---")
    print(f" {board[3]} | {board[4]} | {board[5]} ")
    print("---+---+---")
    print(f" {board[6]} | {board[7]} | {board[8]} ")

# Create a function to check if anyone won, Use marks "X" or "O"
def check_win(player_mark, board):
    win = [f'{player_mark}' * 3]
    return board[:3] == win or board[3:6] == win or board[6:9] == win or \
        [board[0], board[4], board[8]] == win or [board[2], board[4], board[6]] == win or \
        [board[0], board[3], board[6]] == win or [board[1], board[4], board[7]] == win or \
        [board[2], board[5], board[8]] == win

def check_draw(board):
    return ' ' not in board

# Create a Function that makes a copy of the board
def board_copy(board):
    new_board = []
```

```

    for c in board:
        new_board += c
    return new_board

def test_win_move(move, player_mark, board):
    copy = board_copy(board)
    copy[move] = player_mark
    return check_win(player_mark, copy)

def win_strategy(board):
    if board[4] == ' ':
        return 4
    for i in [0, 2, 6, 8]:
        if board[i] == ' ':
            return i
    for i in [1, 3, 5, 7]:
        if board[i] == ' ':
            return i

def get_agent_move(board):
    for i in range(9):
        if board[i] == ' ' and test_win_move(i, 'X', board):
            return i
    for i in range(9):
        if board[i] == ' ' and test_win_move(i, 'O', board):
            return i
    return win_strategy(board)

def tictactoe():
    playing = True

```

```

while playing:
    in_game = True
    board = [' '] * 9
    print('Would you like to go first or second? (1/2)')
    choice = input()
    player_marker = 'O' if choice == '1' else 'X'
    display_board(board)

```

```

while in_game:
    print('\n')
    if player_marker == 'O':
        print('Player move: (0-8)')
        move = int(input())
        if board[move] != ' ':
            print('Invalid move')
            continue
    else:
        move = get_agent_move(board)
    board[move] = player_marker
    if check_win(player_marker, board):
        in_game = False
        display_board(board)
        if player_marker == 'O':
            print('O won')
        else:
            print('X won')
        break
    if check_draw(board):
        in_game = False
        display_board(board)

```

```

        print('The game was a draw.')

        break

    display_board(board)

    if player_marker == 'O':

        player_marker = 'X'

    else:

        player_marker = 'O'

    print('Continue playing? (y/n)')

    ans = input()

    if ans not in 'yY':

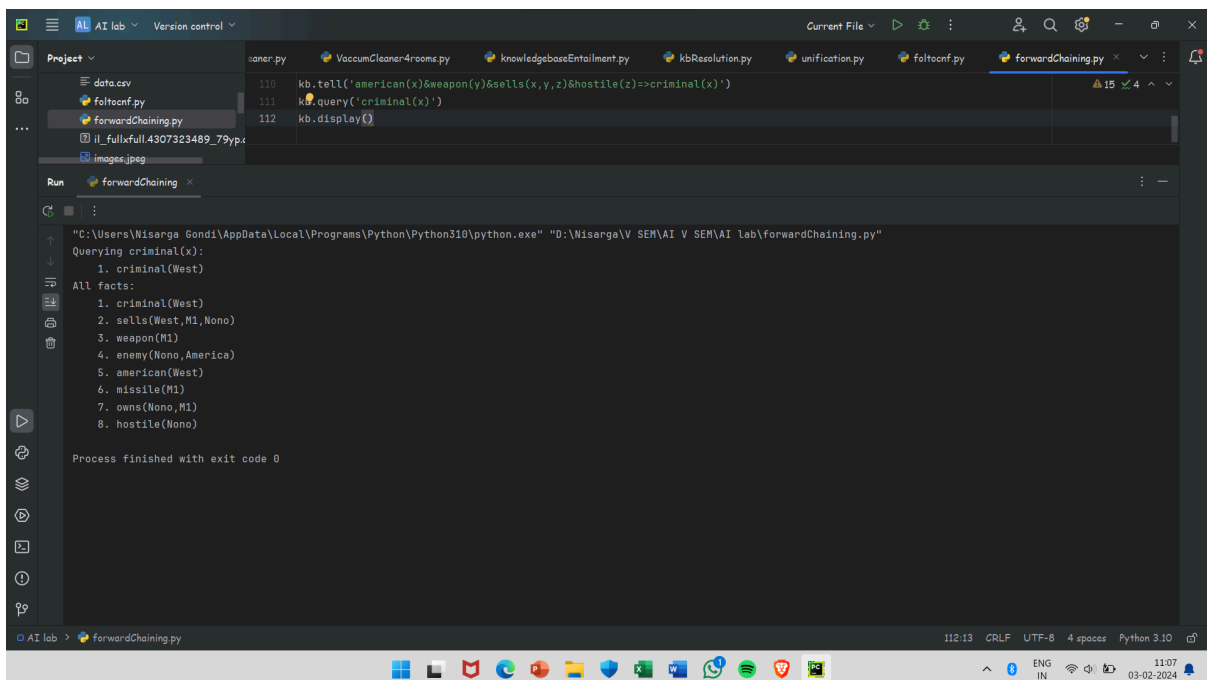
        playing = False

# Play!!!

tictactoe()

```

Output



The screenshot shows a VS Code editor with a project named 'AI lab'. The file explorer on the left shows several files, including 'forwardChaining.py'. The editor window displays the following code in 'forwardChaining.py':

```

110 kb.tell('american(x)&weapon(y)&sells(x,y,z)&hostile(z)=>criminal(x)')
111 kb.query('criminal(x)')
112 kb.display()

```

The Run console at the bottom shows the output of the script:

```

"C:\Users\Wisarga Gondi\AppData\Local\Programs\Python\Python310\python.exe" "D:\Wisarga\AI V SEM\AI lab\forwardChaining.py"
Querying criminal(x):
1. criminal(West)
All facts:
1. criminal(West)
2. sells(West,M1,Nono)
3. weapon(M1)
4. enemy(Nono,America)
5. american(West)
6. missile(M1)
7. owns(Nono,M1)
8. hostile(Nono)
Process finished with exit code 0

```

The status bar at the bottom indicates the file is 'forwardChaining.py' in the 'AI lab' project, using Python 3.10, with a CRLF line ending, UTF-8 encoding, and 4 spaces for indentation. The system clock shows 11:07 on 03-02-2024.

The screenshot shows a Visual Studio Code editor window with a file named `tictactoe.py` open. The editor is in dark theme. The file content is as follows:

```
0 | 0 | X
-----
| X |
-----
| |

Player move: (0-8)
3
0 | 0 | X
-----
0 | X |
-----
| |

0 | 0 | X
-----
0 | X |
-----
X | |
X won
Continue playing? (y/n)
n

Process finished with exit code 0
```

The output of the program is visible in the console at the bottom of the editor. The status bar at the bottom indicates the file is in the `AI lab` workspace, using `Python 3.10` interpreter, with settings `111.1 CRLF UTF-8 4 spaces`. The system tray at the bottom right shows the date and time as `19:26 02-02-2024`.

Program 2

8 Puzzle Breadth First Search Algorithm

Code

```
#import numpy as np
#import pandas as pd
import os

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

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

    return temp # Return the modified state

def possible_moves(state, visited_states):
    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')
```

```

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

pos_moves_it_can = []

for i in d:
    pos_moves_it_can.append(gen(state, i, b))

return [move_it_can for move_it_can in pos_moves_it_can if move_it_can not in
visited_states]

def bfs(src, target):
    queue = []
    queue.append(src)

    exp = []

    while len(queue) > 0:
        source = queue.pop(0)
        exp.append(source)

        print(source[0], '|', source[1], '|', source[2])
        print(source[3], '|', source[4], '|', source[5])
        print(source[6], '|', source[7], '|', source[8])
        print()

        if source == target:
            print("success")
            return

```

```
poss_moves_to_do = possible_moves(source, exp)
```

```
for move in poss_moves_to_do:
```

```
    if move not in exp and move not in queue:
```

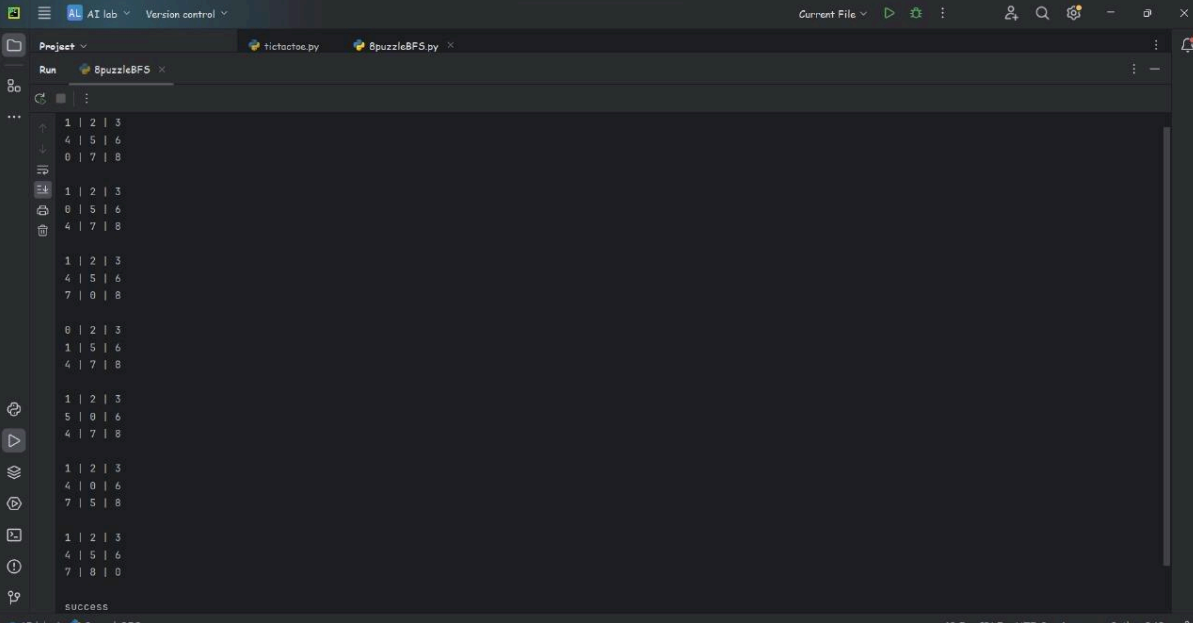
```
        queue.append(move)
```

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

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

```
bfs(src, target)
```

Output



```
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 3

8 Puzzle Iterative deepening search algorithm

Code

```
def id_dfs(puzzle, goal, get_moves):
    import itertools
    #get_moves -> possible_moves

    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 + [move], depth - 1)
                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) # ) indicates White space -> so b has index of it.
    d = [] # direction

    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')
```

```
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:

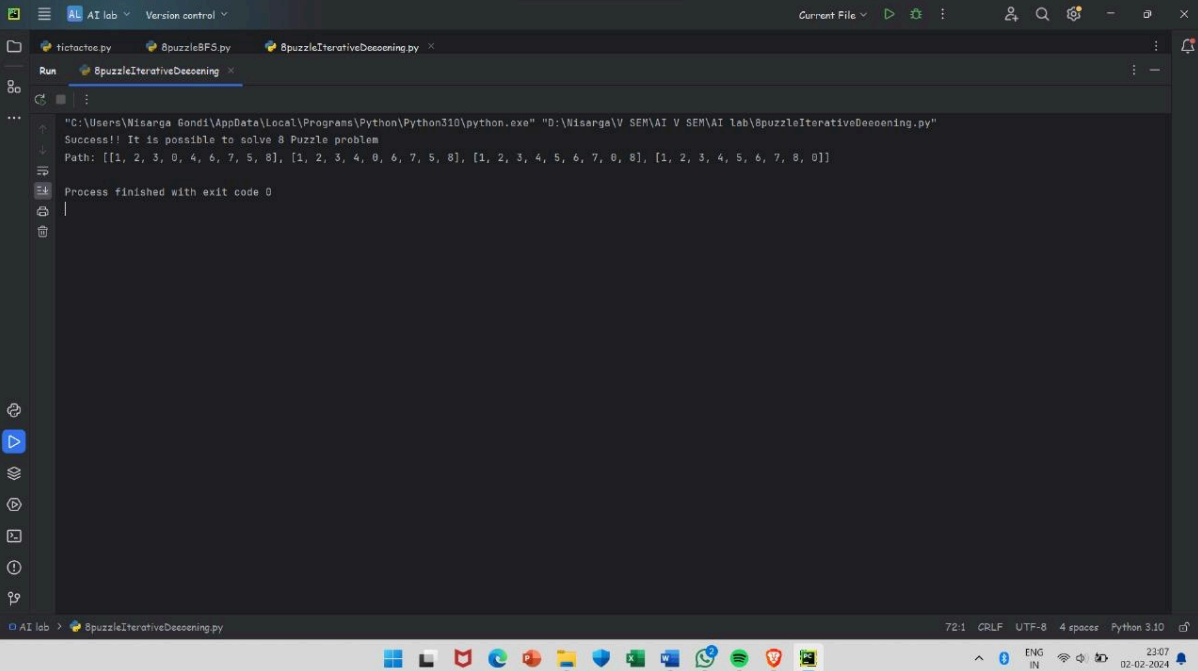
```
print("Success!! It is possible to solve 8 Puzzle problem")
```

```
print("Path:", route)
```

else:

```
print("Failed to find a solution")
```

Output



```
"C:\Users\Nisarga Gondi\AppData\Local\Programs\Python\Python310\python.exe" "D:\Nisarga\V SEM\AI V SEM\AI lab\8puzzleIterativeDecoding.py"
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]]
Process finished with exit code 0
```

Program 4

8 Puzzle A* algorithm

Code

class Node:

```
def __init__(self,data,level,fval):
```

```
    """ Initialize the node with the data, level of the node and the calculated fvalue """
```

```
    self.data = data
```

```
    self.level = level
```

```
    self.fval = fval
```

```
def generate_child(self):
```

```
    """ Generate child nodes from the given node by moving the blank space
```

```
        either in the four directions {up,down,left,right} """
```

```
    x,y = self.find(self.data,'_')
```

```
    """ val_list contains position values for moving the blank space in either of
```

```
        the 4 directions [up,down,left,right] respectively. """
```

```
    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+1,0)
```

```
            children.append(child_node)
```

```
    return children
```

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

```
    """ Move the blank space in the given direction and if the position value are out
        of limits the return None """
```

```
    if x2 >= 0 and x2 < len(self.data) and y2 >= 0 and y2 < len(self.data):
```

```
        temp_puz = []
```



```

temp_puz = self.copy(puz)
temp = temp_puz[x2][y2]
temp_puz[x2][y2] = temp_puz[x1][y1]
temp_puz[x1][y1] = temp
return temp_puz
else:
    return None

```

```

def copy(self,root):
    """ Copy function to create a similar matrix of the given node """
    temp = []
    for i in root:
        t = []
        for j in i:
            t.append(j)
        temp.append(t)
    return temp

```

```

def find(self,puz,x):
    """ Specifically used to find the position of the blank space """
    for i in range(0,len(self.data)):
        for j in range(0,len(self.data)):
            if puz[i][j] == x:
                return i,j

```

```

class Puzzle:
    def __init__(self,size):
        """ Initialize the puzzle size by the specified size,open and closed lists to empty """
        self.n = size

```

```

self.open = []
self.closed = []

def accept(self):
    """ Accepts the puzzle from the user """
    puz = []
    for i in range(0,self.n):
        temp = input().split(" ")
        puz.append(temp)
    return puz

def f(self,start,goal):
    """ Heuristic Function to calculate hueristic value  $f(x) = h(x) + g(x)$  """
    return self.h(start.data,goal)+start.level

def h(self,start,goal):
    """ Calculates the different between the given puzzles """
    temp = 0
    for i in range(0,self.n):
        for j in range(0,self.n):
            if start[i][j] != goal[i][j] and start[i][j] != '_':
                temp += 1
    return temp

def process(self):
    """ Accept Start and Goal Puzzle state"""
    print("Enter the start state matrix \n")
    start = self.accept()
    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"""
self.open.append(start)
print("\n\n")
while True:
    cur = self.open[0]
    print("")
    print(" | ")
    print(" | ")
    print(" \\/ \n")
    for i in cur.data:
        for j in i:
            print(j,end=" ")
        print("")
    """ If the difference between current and goal node is 0 we have reached the goal
node"""
    if(self.h(cur.data,goal) == 0):
        break
    for i in cur.generate_child():
        i.fval = self.f(i,goal)
        self.open.append(i)
    self.closed.append(cur)
    del self.open[0]

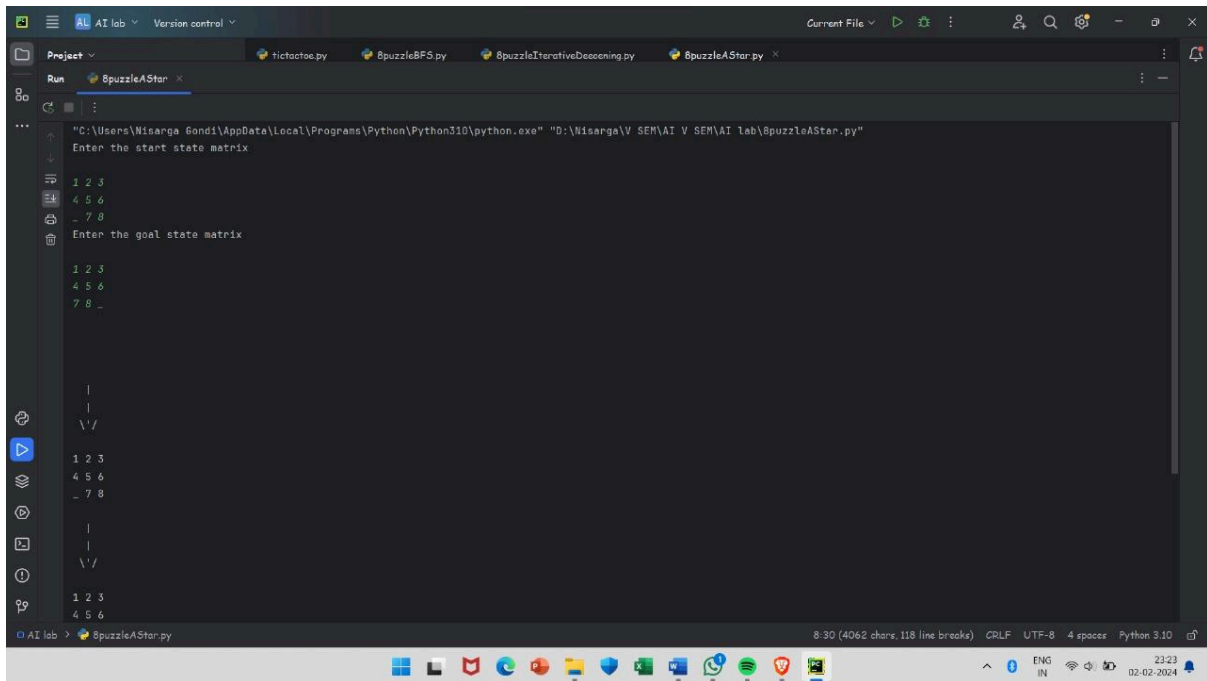
    """ sort the opne list based on f value """
    self.open.sort(key = lambda x:x.fval,reverse=False)

```

```
puz = Puzzle(3)
```

```
puz.process()
```

Output



The screenshot shows a Python IDE window titled "AI lab" with a "Version control" dropdown. The "Run" tab is active, displaying the execution of a script named "8puzzleAStar.py". The command line shows the script being run from the directory "D:\Nisarga\AI V SEM\AI lab\8puzzleAStar.py". The program prompts the user to "Enter the start state matrix" and "Enter the goal state matrix". The start state matrix is entered as:

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

The goal state matrix is entered as:

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

The program then displays the solution path, showing the sequence of moves: "1", "1", and "V/V". The final state of the puzzle is shown as:

```
1 2 3
4 5 6
```

The status bar at the bottom indicates the file is 8:30 (4062 chars, 118 line breaks), CRLF, UTF-8, 4 spaces, and Python 3.10. The system tray shows the time as 23:23 on 02-02-2024.

Program 5

Vacuum Cleaner

Code

```
def clean_room(room_name, is_dirty):
    if is_dirty:
        print(f"Cleaning {room_name} (Room was dirty)")
        print(f"{room_name} is now clean.")
        return 0 # Updated status after cleaning
    else:
        print(f"{room_name} is already clean.")
        return 0 # Status remains clean

def main():
    rooms = ["Room 1", "Room 2"]
    room_statuses = []

    for room in rooms:
        status = int(input(f"Enter clean status for {room} (1 for dirty, 0 for clean): "))
        room_statuses.append((room, status))
    print(room_statuses)

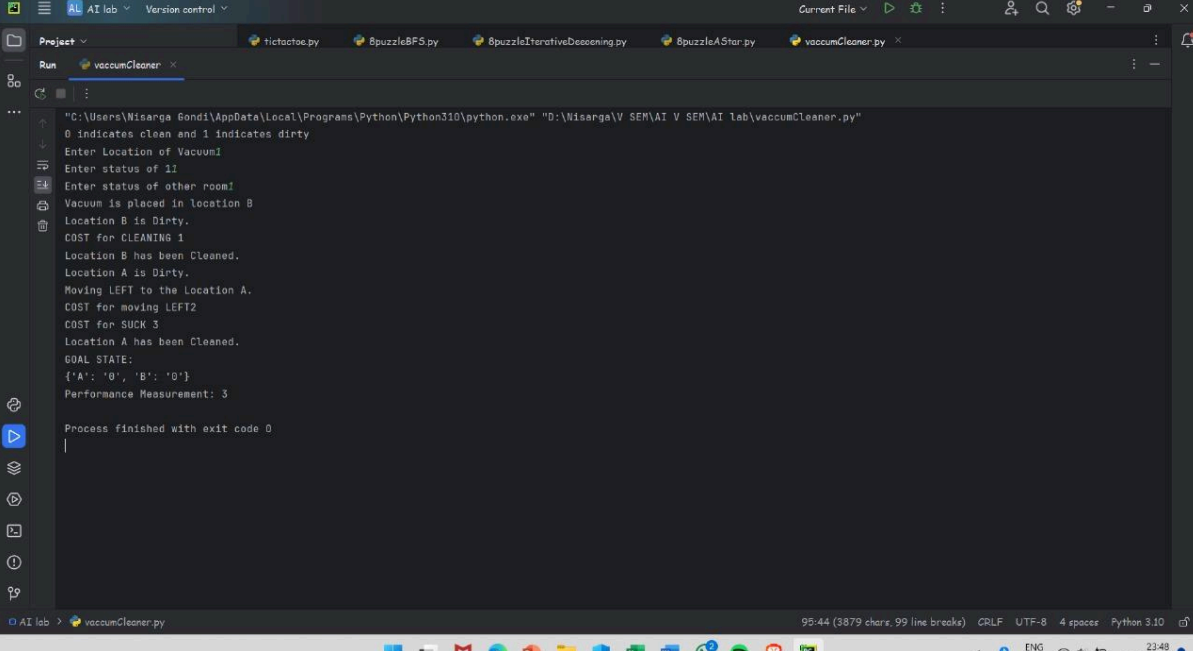
    for i, (room, status) in enumerate(room_statuses):
        room_statuses[i] = (room, clean_room(room, status)) # Update status after cleaning

    print(f"Returning to {rooms[0]} to check if it has become dirty again:")
    room_statuses[0] = (rooms[0], clean_room(rooms[0], room_statuses[0][1])) # Checking
    Room 1 after cleaning all rooms

    print(f"{rooms[0]} is {'dirty' if room_statuses[0][1] else 'clean'} after checking.")
```

```
if __name__ == "__main__":  
    main()
```

Output



```
"C:\Users\Wisarga Gondi\AppData\Local\Programs\Python\Python310\python.exe" "D:\Wisarga\V SEM\AI V SEM\AI lab\vacuumCleaner.py"  
0 indicates clean and 1 indicates dirty  
Enter Location of Vacuum1  
Enter status of 11  
Enter status of other room1  
Vacuum is placed in location 0  
Location B is Dirty.  
COST for CLEANING 1  
Location B has been Cleaned.  
Location A is Dirty.  
Moving LEFT to the Location A.  
COST for moving LEFT2  
COST for SUCK 3  
Location A has been Cleaned.  
GOAL STATE:  
{'A': '0', 'B': '0'}  
Performance Measurement: 3  
  
Process finished with exit code 0
```

Program 6

Knowledge base entailment

Code

```
from sympy import symbols, And, Not, Implies, satisfiable

def create_knowledge_base():
    # Define propositional symbols
    p = symbols('p')
    q = symbols('q')
    r = symbols('r')

    # Define knowledge base using logical statements
    knowledge_base = And(
        Implies(p, q), # If p then q
        Implies(q, r), # If q then r
        Not(r) # Not r
    )

    return knowledge_base

def query_entails(knowledge_base, query):
    # Check if the knowledge base entails the query
    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()

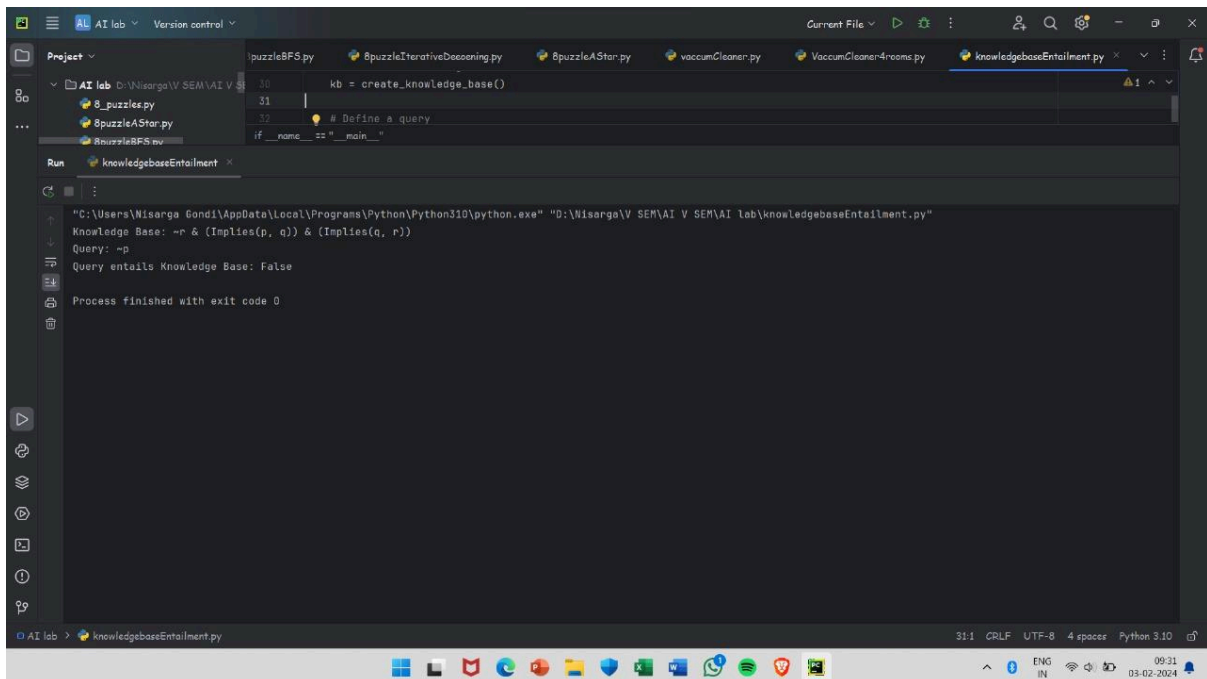
# Define a query
query = symbols('p')

# 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)

```

Output



```

C:\Users\Nisarga Gondli\AppData\Local\Programs\Python\Python310\python.exe "D:\Nisarga\AI lab\knowledgebaseEntailment.py"
Knowledge Base: ~r & (Implies(p, q)) & (Implies(q, r))
Query: ~p
Query entails Knowledge Base: False
Process finished with exit code 0

```


Program 7

Knowledge base resolution

Code

```
import re

def main(rules, goal):
    rules = rules.split(' ')
    steps = resolve(rules, goal)
    print("\nStep\t|Clause\t|Derivation\t")
    print('-' * 30)
    i = 1
    for step in steps:
        print(f' {i}.\t| {step}\t| {steps[step]}\t')
        i += 1

def negate(term):
    return f'~{term}' if term[0] != '~' else term[1]

def reverse(clause):
    if len(clause) > 2:
        t = split_terms(clause)
        return f'{t[1]}v{t[0]}'
    return ""

def split_terms(rule):
    exp = '(~*[PQRS])'
    terms = re.findall(exp, rule)
    return terms

split_terms('~PvR')
```

```

def contradiction(goal, clause):
    contradictions = [ f'{goal} v {negate(goal)}', f'{negate(goal)} v {goal}' ]
    return clause in contradictions or reverse(clause) in contradictions

def resolve(rules, goal):
    temp = rules.copy()
    temp += [negate(goal)]
    steps = dict()
    for rule in temp:
        steps[rule] = 'Given.'
    steps[negate(goal)] = 'Negated conclusion.'
    i = 0
    while i < len(temp):
        n = len(temp)
        j = (i + 1) % n
        clauses = []
        while j != i:
            terms1 = split_terms(temp[i])
            terms2 = split_terms(temp[j])
            for c in terms1:
                if negate(c) in terms2:
                    t1 = [t for t in terms1 if t != c]
                    t2 = [t for t in terms2 if t != negate(c)]
                    gen = t1 + t2
                    if len(gen) == 2:
                        if gen[0] != negate(gen[1]):
                            clauses += [f'{gen[0]} v {gen[1]}']
                        else:
                            if contradiction(goal, f'{gen[0]} v {gen[1]}'):

```

```

temp.append(f'{gen[0]}v{gen[1]}')
steps[""] = f'Resolved {temp[i]} and {temp[j]} to {temp[-1]}, which is in
turn null. \

```

\nA contradiction is found when {negate(goal)} is assumed as true.
Hence, {goal} is true."

```

return steps
elif len(gen) == 1:
    clauses += [f'{gen[0]}']
else:
    if contradiction(goal,f'{terms1[0]}v{terms2[0]}'):
        temp.append(f'{terms1[0]}v{terms2[0]}')
        steps[""] = f'Resolved {temp[i]} and {temp[j]} to {temp[-1]}, which is in
turn null. \

```

\nA contradiction is found when {negate(goal)} is assumed as true. Hence,
{goal} is true."

```

return steps
for clause in clauses:
    if clause not in temp and clause != reverse(clause) and reverse(clause) not in temp:
        temp.append(clause)
        steps[clause] = f'Resolved from {temp[i]} and {temp[j]}.'
    j = (j + 1) % n
    i += 1
return steps

```

```

rules = 'Rv~P Rv~Q ~RvP ~RvQ' #(P^Q)<=>R : (Rv~P)v(Rv~Q)^(~RvP)^(~RvQ)

```

```

goal = 'R'

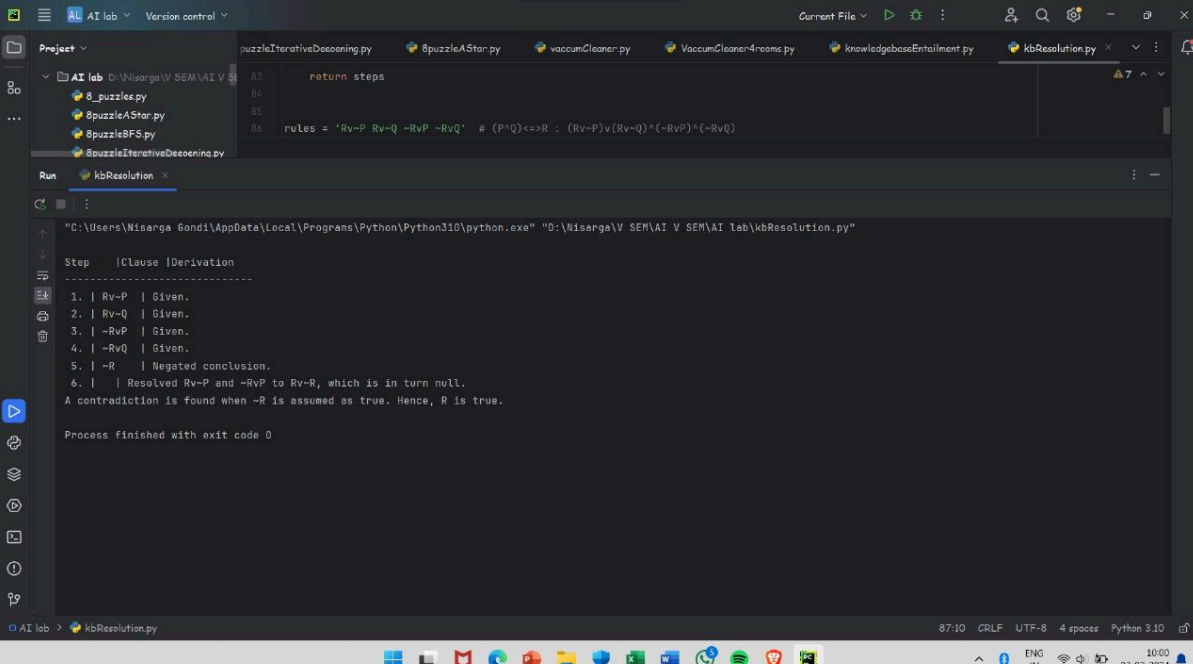
```

```

main(rules, goal)

```

Output



The screenshot shows a code editor with a project named "AI lab" and a file named "kbResolution.py". The script contains a function that returns steps and a set of rules. The output of the script is displayed in the console, showing a table of steps and a final conclusion.

```
83     return steps
84
85
86 rules = 'Rv~P Rv~Q ~RvP ~RvQ' # (P^Q)<=>R : (Rv~P)v(Rv~Q)^(~RvP)^(~RvQ)
```

Run kbResolution

"C:\Users\Wisarga Gond\AppData\Local\Programs\Python\Python310\python.exe" "D:\Wisarga\AI lab\kbResolution.py"

Step	Clause	Derivation
1.	Rv~P	Given.
2.	Rv~Q	Given.
3.	~RvP	Given.
4.	~RvQ	Given.
5.	~R	Negated conclusion.
6.		Resolved Rv~P and ~RvP to Rv~R, which is in turn null.

A contradiction is found when ~R is assumed as true. Hence, R is true.

Process finished with exit code 0

Program 8

Unification

Code

```
import re

def getAttributes(expression):
    expression = expression.split("(")[1:]
    expression = "(" .join(expression)
    expression = expression[:-1]
    expression = re.split("(?<!\(.\)(?!.\))", expression)
    return expression

def getInitialPredicate(expression):
    return expression.split("(")[0]

def isConstant(char):
    return char.isupper() and len(char) == 1

def isVariable(char):
    return char.islower() 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 checkOccurs(var, exp):
    if exp.find(var) == -1:
        return False
    return True

```

```

def getFirstPart(expression):
    attributes = getAttributes(expression)
    return attributes[0]

```

```

def getRemainingPart(expression):
    predicate = getInitialPredicate(expression)
    attributes = getAttributes(expression)
    newExpression = predicate + "(" + ",".join(attributes[1:]) + ")"
    return newExpression

```

```

def unify(exp1, exp2):
    if exp1 == exp2:
        return []

    if isConstant(exp1) and isConstant(exp2):
        if exp1 != exp2:
            return False

```

```
if isConstant(exp1):  
    return [(exp1, exp2)]
```

```
if isConstant(exp2):  
    return [(exp2, exp1)]
```

```
if isVariable(exp1):  
    if checkOccurs(exp1, exp2):  
        return False  
    else:  
        return [(exp2, exp1)]
```

```
if isVariable(exp2):  
    if checkOccurs(exp2, exp1):  
        return False  
    else:  
        return [(exp1, exp2)]
```

```
if getInitialPredicate(exp1) != getInitialPredicate(exp2):  
    print("Predicates do not match. Cannot be unified")  
    return False
```

```
attributeCount1 = len(getAttributes(exp1))  
attributeCount2 = len(getAttributes(exp2))  
if attributeCount1 != attributeCount2:  
    return False
```

```
head1 = getFirstPart(exp1)  
head2 = getFirstPart(exp2)  
initialSubstitution = unify(head1, head2)
```

```

if not initialSubstitution:
    return False
if attributeCount1 == 1:
    return initialSubstitution

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

if initialSubstitution != []:
    tail1 = apply(tail1, initialSubstitution)
    tail2 = apply(tail2, initialSubstitution)

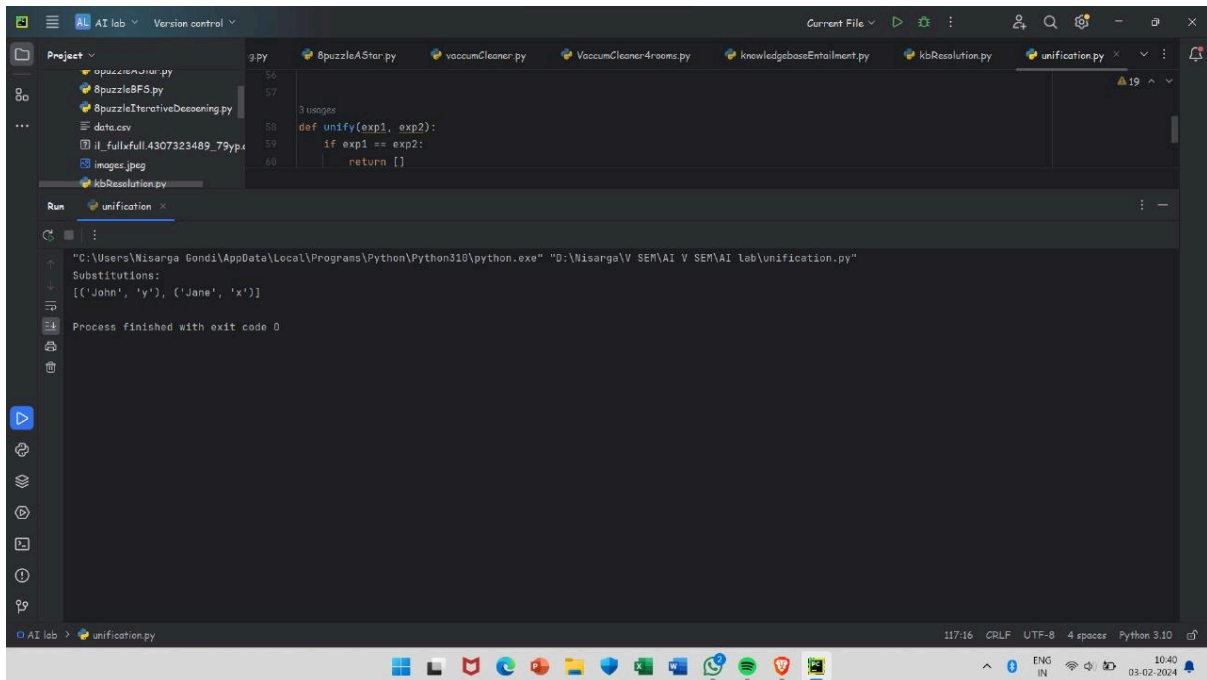
remainingSubstitution = unify(tail1, tail2)
if not remainingSubstitution:
    return False

initialSubstitution.extend(remainingSubstitution)
return initialSubstitution

exp1 = "knows(A,x)"
exp2 = "knows(y,Y)"
substitutions = unify(exp1, exp2)
print("Substitutions:")
print(substitutions)

```


Output



The screenshot shows a code editor with a project named 'AI lab'. The file 'unification.py' is open, showing a function 'unify' that takes two expressions and returns a list of substitutions. The output window shows the execution of the script, displaying the path to the Python interpreter, the substitutions found, and the exit code.

```
def unify(exp1, exp2):  
    if exp1 == exp2:  
        return []
```

Run unification

"C:\Users\Nisarga Gondil\AppData\Local\Programs\Python\Python310\python.exe" "D:\Nisarga\SEM\AI V SEM\AI lab\unification.py"

Substitutions:
[('John', 'y'), ('Jane', 'x')]

Process finished with exit code 0

Program 9

FOL to CNF

Code

```
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 Skolemization(statement):
    SKOLEM_CONSTANTS = [f'{chr(c)}' for c in range(ord('A'), ord('Z')+1)]
    matches = re.findall('[\exists].', statement)
    for match in matches[::-1]:
        statement = statement.replace(match, "")
        for predicate in getPredicates(statement):
            attributes = getAttributes(predicate)
            if ".join(attributes).islower()":
                statement = statement.replace(match[1], SKOLEM_CONSTANTS.pop(0))
    return statement

import re

def fol_to_cnf(fol):
    statement = fol.replace("=>", "-")
    expr = '\([^)]+\)'
    statements = re.findall(expr, statement)
    for i, s in enumerate(statements):
```

```

    if '[' in s and ']' not in s:
        statements[i] += ']'
for s in statements:
    statement = statement.replace(s, fol_to_cnf(s))
while '-' in statement:
    i = statement.index('-')
    br = statement.index('[') if '[' in statement else 0
    new_statement = '~' + statement[br:i] + '|' + statement[i+1:]
    statement = statement[:br] + new_statement if br > 0 else new_statement
return Skolemization(statement)

print(fol_to_cnf("bird(x)=>~fly(x)"))
print(fol_to_cnf("∃x[bird(x)=>~fly(x)]"))

```

Output

```

C:\Users\Nisarga Gondi\AppData\Local\Programs\Python\Python310\python.exe "D:\Nisarga\V SEM\AI V SEM\AI lab\foltocnf.py"
~bird(x)|~fly(x)
[~bird(A)|~fly(A)]
[~bird(A)|~fly(A)]
Process finished with exit code 0

```

Program 10

Forward Chaining

Code

```
import re

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

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

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

class Fact:
    def __init__(self, expression):
        self.expression = expression
        predicate, params = self.splitExpression(expression)
        self.predicate = predicate
        self.params = params
        self.result = any(self.getConstants())

    def splitExpression(self, expression):
        predicate = getPredicates(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 getVariables(self):
    return [v if isVariable(v) else None for v in self.params]

def substitute(self, constants):
    c = constants.copy()
    f = f"{self.predicate}({''.join([constants.pop(0) if isVariable(p) else p for p in
self.params]))}"
    return Fact(f)

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.getVariables()):
                        if v:
                            constants[v] = fact.getConstants()[i]

```

```

        new_lhs.append(fact)

    predicate, attributes = getPredicates(self.rhs.expression)[0],
str(getAttributes(self.rhs.expression)[0])

    for key in constants:
        if constants[key]:
            attributes = attributes.replace(key, constants[key])

    expr = f' {predicate} {attributes}'

    return Fact(expr) if len(new_lhs) and all([f.getResult() for f in new_lhs]) else None

```

```

class KB:

```

```

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

    def tell(self, e):
        if '=>' in e:
            self.implications.add(Implication(e))
        else:
            self.facts.add(Fact(e))

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

```

```

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

        i = 1
        print(f'Querying {e}:')

        for f in facts:
            if Fact(f).predicate == Fact(e).predicate:
                print(f'\t{i}. {f}')

```

```
i += 1
```

```
def display(self):  
    print("All facts: ")  
    for i, f in enumerate(set([f.expression for f in self.facts])):  
        print(f'\t{i+1}. {f}')
```

```
kb = KB()  
kb.tell('missile(x)=>weapon(x)')  
kb.tell('missile(M1)')  
kb.tell('enemy(x,America)=>hostile(x)')  
kb.tell('american(West)')  
kb.tell('enemy(Nono,America)')  
kb.tell('owns(Nono,M1)')  
kb.tell('missile(x)&owns(Nono,x)=>sells(West,x,Nono)')  
kb.tell('american(x)&weapon(y)&sells(x,y,z)&hostile(z)=>criminal(x)')  
kb.query('criminal(x)')  
kb.display()
```

Output

```
Project: AI lab  
- date.csv  
- foltoconf.py  
- forwardChaining.py  
- I_fullfull.4307323489_79ypa  
- images.jpeg  
- ...  
Run: forwardChaining  
"C:\Users\Wisarga Gondil\AppData\Local\Programs\Python\Python310\python.exe" "D:\Misarga\AI V SEM\AI Lab\forwardChaining.py"  
Querying criminal(x):  
1. criminal(West)  
All facts:  
1. criminal(West)  
2. sells(West,M1,Nono)  
3. weapon(M1)  
4. enemy(Nono,America)  
5. american(West)  
6. missile(M1)  
7. owns(Nono,M1)  
8. hostile(Nono)  
Process finished with exit code 0
```

