VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.



LAB REPORT on

Artificial Intelligence

Submitted by

VAIBHAV KIRAN P (1BM21CS233)

in partial fulfillment 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
Nov-2023 to Feb-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 VAIBHAV KIRAN P (1BM21CS233), who is bonafide student of B.M.S. College of Engineering. It is in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the academic semester June-2023 to Sep-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a Artificial Intelligence (22CS5PCAIN) work prescribed for the said degree.

Sandhya A Kulkarni	Dr. Jyothi S Nayak

Assistant Professor Professor and Head

Department of CSE

BMSCE, Bengaluru

BMSCE, Bengaluru

Index Sheet

Lab Program No.	Program Details			
1	Implement Tic –Tac –Toe Game.	1 - 10		
2	Solve 8 puzzle problems.	11 - 17		
3	Implement Iterative deepening search algorithm.	18 - 24		
4	Implement A* search algorithm.	25 – 31		
5	Implement vaccum cleaner agent.			
6	Create a knowledge base using prepositional logic and show that the given query entails the knowledge base or not.	38 – 41		
7	Create a knowledge base using prepositional logic and prove the given query using resolution			
8	Implement unification in first order logic	49 – 55		
9	Convert a given first order logic statement into Conjunctive Normal Form (CNF).	56 – 59		
10	Create a knowledge base consisting of first order logic statements and prove the given query using forward reasoning.	60 - 67		

Course Outcome

CO1	Apply knowledge of agent architecture, searching and reasoning techniques for different applications.
CO2	Analyse Searching and Inferencing Techniques.
CO3	Design a reasoning system for a given requirement.
CO4	Conduct practical experiments for demonstrating agents, searching and inferencing.

1. Implement Tic -Tac -Toe Game.

```
import moth
import copy
 0 = 0
 EMPTY = None
de quital_state ();
   SLETTEN [[EMPTY, EMPTY, EMPTY],
          [EMPTY, EMPTY, EMPTY]
          [EMPTY, EMPTY, EMPTY]]
ole player (board);
  count 0=0
   (ount x = 0
   for y in [0,1,2];
       for x in 600ad (y];
        Count 0 = Count 0+1
         else i x == X"
         count x = count x + 1
       count 0 >= count x;
            estien X (1) 11 back bearings to 1/20
       elseif count x > count 0:
            setun o
     del actions (board):
         free boxee = set()
          for in [0,1,2]:
             bor j ?n [0,1.2]:
                ? board [i][j] = EMPTY:
                   free Loxer add ((i,j))
          getuen prectoxes
       dy swult (board, action):
              " = autem [0]
```

```
j = action [1]
       If type (aution) = = list:
           action = (i, i)
"I action in actions (board):
       ? | player (bound) == x: Loard [i] = x
   elie if player (board) = = 10;
         0 = [1][1] bood
outurn Loand
  winner (board):
  16 (boosed (0)[0] == 6000d [0][1] ==
      500ad [0][2] == x or 500ad [1][0]
     == board [i][i] == Lowed [i][2]==X
 or board [2][0] == board [2][1] ==
     board [2][2] == X):
     x neutron X
  (same as above == 0):
       situan 0
  for i in [0,1,2]:
      82=[]
       for 1 in [0,1,2]!
       82 appard (board [j][i])
        (S2[0]== S2[1]== S1[2]):
          relien 82 [0]
   Stike D = []
   for ? in Co,1,21:
     Stoke D. apprand (Sound [i][])
 91 ( Strike Dlo'T = = strike Dli] = = strike D (2)):
         seturn strike D Po]
    1/ (boosd Poll > 1 = 500ord (1)[1] >= 500ord (2)(0)):
      sulturn Social CoJ[2]
       suturn None
   def terminal (board):
            Full = True
          for 9 in Co11,21:
             for j in board [i]:
               ? | is Mone;
                   T-ull > Falls
            il Full!
```

```
Juturn Tree
   if ( usiner (board) in not mone):
   Juturn True
  seturn False
   utility (600901):
def
      ? | ( winner ( board) == 0 : )
           setum-1 and board small still
       esse:
         section 0
    nevimas - helper (board);
    is maxTurn = True if player (board) == x
                   ele Falie
   if terminal (board): ( Least tracked of ) the
      seturn utility (board)
      8 core = []
   for move in actions (board):
      Qualit (board, 1900)
      Scorel. append (minnax helper (Load))
      board [move 807] thore 81] = EMPTY
    Suturn man ( Scores) if israx Temele nun (Scores)
  de minimax (60000):
     is max Ton = True of player (boord) = = x
                                  else False
    Leastmore = None
   Ph is Max Torn:
         best score = - math. Pnf
        for move in outline (board):
               escult (board . move)
               Blore = minimax . helper (board)
               board (more [0]] [more [1]] = ETIPTY
         if (score > beatscore):
                 Lest score = Score
                  best More = move
           return belt More
      elle:
```

```
while not terminal (game-board):
        if player (game-board) == X!
           User-Enput = Enput (11/n Enteryour move (now. column):)
           900.col = map (Port, wer-input spat (:))
            swell (game - board, (row.col))
        else:
         pont ("In Al " making amove...")
         Move = Minimax (Copy: deopospy ( game - board);
           swell (game - so and , move)
         print ("In Cusent Board:")
         print - board (game - board)
ib minner (game boord) is not wone:
      print ("In the winner is: { winner (game board) 34);
              ones append (without liber Heads)
          print ("In It's atie!")
 output :-
 Entry your move (1-9):3
  Enter your move (1-9):1
                            Entery our move (1-9): 7
   Enter your move (1-9): 2
                                player x wins ?
   Enter you move (1-9): 4
```

Implement Tic -Tac -Toe Game.

```
import math
import copy
X = "X"
O = "O"
EMPTY = None
def initial_state():
  return [[EMPTY, EMPTY, EMPTY],
      [EMPTY, EMPTY, EMPTY],
      [EMPTY, EMPTY, EMPTY]]
def player(board):
  countO = 0
  countX = 0
  for y in [0, 1, 2]:
    for x in board[y]:
      if x == "O":
         countO = countO + 1
      elif x == "X":
         countX = countX + 1
  if countO >= countX:
    return X
  elif countX > countO:
    return O
```

```
def actions(board):
  freeboxes = set()
  for i in [0, 1, 2]:
     for j in [0, 1, 2]:
       if board[i][j] == EMPTY:
          freeboxes.add((i, j))
  return freeboxes
def result(board, action):
  i = action[0]
  j = action[1]
  if type(action) == list:
     action = (i, j)
  if action in actions(board):
     if player(board) == X:
       board[i][j] = X
     elif player(board) == O:
       board[i][j] = O
  return board
def winner(board):
  if (board[0][0] == board[0][1] == board[0][2] == X or board[1][0] == board[1][1] ==
board[1][2] == X \text{ or } board[2][0] == board[2][1] == board[2][2] == X):
     return X
  if (board[0][0] == board[0][1] == board[0][2] == O or board[1][0] == board[1][1] ==
board[1][2] == O \text{ or } board[2][0] == board[2][1] == board[2][2] == O):
     return O
  for i in [0, 1, 2]:
     s2 = []
```

```
for j in [0, 1, 2]:
       s2.append(board[j][i])
     if (s2[0] == s2[1] == s2[2]):
       return s2[0]
  strikeD = []
  for i in [0, 1, 2]:
     strikeD.append(board[i][i])
  if (strikeD[0] == strikeD[1] == strikeD[2]):
     return strikeD[0]
  if (board[0][2] == board[1][1] == board[2][0]):
     return board[0][2]
  return None
def terminal(board):
  Full = True
  for i in [0, 1, 2]:
     for j in board[i]:
       if j is None:
          Full = False
  if Full:
     return True
  if (winner(board) is not None):
     return True
  return False
def utility(board):
  if (winner(board) == X):
     return 1
```

```
elif winner(board) == O:
    return -1
  else:
    return 0
def minimax_helper(board):
  isMaxTurn = True if player(board) == X else False
  if terminal(board):
    return utility(board)
  scores = []
  for move in actions(board):
    result(board, move)
    scores.append(minimax_helper(board))
    board[move[0]][move[1]] = EMPTY
  return max(scores) if isMaxTurn else min(scores)
def minimax(board):
  isMaxTurn = True if player(board) == X else False
  bestMove = None
  if isMaxTurn:
    bestScore = -math.inf
    for move in actions(board):
       result(board, move)
       score = minimax_helper(board)
       board[move[0]][move[1]] = EMPTY
       if (score > bestScore):
         bestScore = score
```

```
bestMove = move
    return bestMove
  else:
    bestScore = +math.inf
    for move in actions(board):
       result(board, move)
       score = minimax_helper(board)
       board[move[0]][move[1]] = EMPTY
       if (score < bestScore):
         bestScore = score
         bestMove = move
    return bestMove
def print_board(board):
  for row in board:
    print(row)
# Example usage:
game_board = initial_state()
print("Initial Board:")
print_board(game_board)
while not terminal(game_board):
  if player(game_board) == X:
    user_input = input("\nEnter your move (row, column): ")
    row, col = map(int, user_input.split(','))
    result(game_board, (row, col))
  else:
```

```
print("\nAI is making a move...")
  move = minimax(copy.deepcopy(game_board))
  result(game_board, move)

print("\nCurrent Board:")
  print_board(game_board)

# Determine the winner
if winner(game_board) is not None:
  print(f"\nThe winner is: {winner(game_board)}")
else:
  print("\nIt's a tie!")
```

OUTPUT:

```
Initial Board:
[None, None, None]
[None, None, None]
[None, None, None]
Enter your move (row, column): 1,2
Current Board:
[None, None, None]
[None, None, 'X']
[None, None, None]
AI is making a move...
Current Board:
[None, None, None]
[None, '0', 'X']
[None, None, None]
Enter your move (row, column): 0,0
Current Board:
['X', None, None]
[None, 'O', 'X']
[None, None, None]
AI is making a move...
Current Board:
['X', '0', None]
[None, '0', 'X']
[None, None, None]
Enter your move (row, column): 2,1
```

```
Current Board:
['X', '0', None]
[None, '0', 'X']
[None, 'X', None]

AI is making a move...

Current Board:
['X', '0', None]
[None, '0', 'X']
['0', 'X', None]

Enter your move (row, column): 1,0

Current Board:
['X', '0', None]
['X', '0', None]
['X', '0', 'X']
['0', 'X', None]

AI is making a move...

Current Board:
['X', '0', '0']
['X', '0', '0']
['X', '0', 'X']
['0', 'X', None]

The winner is: 0
```

2. Solve 8 puzzle problems.

```
Puzzle
del 48 (src, taget):
   quare = []
   gieve append (orc)
    exp > ()
   whole In (queue)>0: 1 have beginn
     Bouse = queve pop(0)
     exp append (source)
     print ( source)
     " ( CBOLACE = = +taget ):
    print ("success")
       Jutum
   poer-mover-to-do = []
  poss - mover - to-do = possible - mover (source, exp)
   for move in poss_mover to-do;
      of move not in exp and move not in
              (3537037811) - there
       queve:
    queve_append (move)
 del possible moves ( stade, vierted states):
     b = State. andex(0)
      d= []
     if b not in Coilid]:
         d append ('U')
     16 P mat in [8:18].
        d append (d').
     1 L not in [0,3,6]:
       d-append ('1')
     11 6 not in [2,5,8]:
       d append (Y')
```

```
POB - moves - ?+ - (um = ()
   borin a
     pos-mover-it-can-append (gen (Statue 1, b))
  section Prove - it - can for
        move it - lan in pos-mover -it - lun
     if move -it-un not in
         visited - statell]
del gen [ state, m. b.)
     temp = state · copy ()
        of m = = d'.
        temp [678] .temp [6] : temp [6] , temp [4+18]
     if m = = u'
         temp (6-17, temp (6) - temp (6), temp (6),
                                  temp (b+1)
   gustum temp
Crc = [1,21310,4.5161718]
 tagget = (1,2,3,4,5,0,6,7,8)
Src = [2,0,3,1,8,4,7,6,5]
 taget = (1,2,3,8,0,4,7,6,5]
bla ( src, tagget)
output :-
           Step 2:
                        Step 3:
   Step 1:
             123
                        123
   123
                        458
             458
                                 458
   45-1
                        6-17
            67-1
    678
```

Step 5! 123 458 -167	step 6: 123 5-18 467	step 7: 123 568 4-17	Step 8: 1 2 3 5 6 8 4 7 -1
Step 9: 12 3 86-1 47 8	Sty 10! 123 5-16 478	47 8	Sep12: 123 456 -178
Step13 123 456 7-18	Step14 1 2 3 45 6 7 3 -1		t time type for a type
Maria James	, stantalogue adopre	tine) dress.	had days yes

Solve 8 puzzle problems.

```
def bfs(src,target):
  queue = []
  queue.append(src)
  exp = []
  while len(queue) > 0:
    source = queue.pop(0)
    exp.append(source)
    print(source)
    if source==target:
       print("Success")
       return
    poss_moves_to_do = []
    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)
def possible_moves(state,visited_states):
  #index of empty spot
  b = state.index(0)
```

```
#directions array
  d = []
  #Add all the possible directions
  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')
  # If direction is possible then add state to move
  pos_moves_it_can = []
  # for all possible directions find the state if that move is played
  ### Jump to gen function to generate all possible moves in the given directions
  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 gen(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 new state with tested move to later check if "src == target"
  return temp
print("Example 1")
src=[2,0,3,1,8,4,7,6,5]
target=[1,2,3,8,0,4,7,6,5]
print("Source: " , src)
print("Goal State: " , target)
bfs(src, target)
print("\nExample 2")
src = [1,2,3,0,4,5,6,7,8]
target = [1,2,3,4,5,0,6,7,8]
print("Source: " , src)
print("Goal State: " , target)
bfs(src, target)
```

OUTPUT:

```
Example 1
Source: [2, 0, 3, 1, 8, 4, 7, 6, 5]
Goal State: [1, 2, 3, 8, 0, 4, 7, 6, 5]
[2, 0, 3, 1, 8, 4, 7, 6, 5]
[2, 8, 3, 1, 0, 4, 7, 6, 5]
[0, 2, 3, 1, 8, 4, 7, 6, 5]
[2, 3, 0, 1, 8, 4, 7, 6, 5]
[2, 8, 3, 1, 6, 4, 7, 0, 5]
[2, 8, 3, 0, 1, 4, 7, 6, 5]
[2, 8, 3, 1, 4, 0, 7, 6, 5]
[1, 2, 3, 0, 8, 4, 7, 6, 5]
[2, 3, 4, 1, 8, 0, 7, 6, 5]
[2, 8, 3, 1, 6, 4, 0, 7, 5]
[2, 8, 3, 1, 6, 4, 7, 5, 0]
[0, 8, 3, 2, 1, 4, 7, 6, 5]
[2, 8, 3, 7, 1, 4, 0, 6, 5]
[2, 8, 0, 1, 4, 3, 7, 6, 5]
[2, 8, 3, 1, 4, 5, 7, 6, 0]
[1, 2, 3, 7, 8, 4, 0, 6, 5]
[1, 2, 3, 8, 0, 4, 7, 6, 5]
Success
Example 2
Source: [1, 2, 3, 0, 4, 5, 6, 7, 8]
Goal State: [1, 2, 3, 4, 5, 0, 6, 7, 8]
[1, 2, 3, 0, 4, 5, 6, 7, 8]
[0, 2, 3, 1, 4, 5, 6, 7, 8]
[1, 2, 3, 6, 4, 5, 0, 7, 8]
[1, 2, 3, 4, 0, 5, 6, 7, 8]
[2, 0, 3, 1, 4, 5, 6, 7, 8]
[1, 2, 3, 6, 4, 5, 7, 0, 8]
[1, 0, 3, 4, 2, 5, 6, 7, 8]
[1, 2, 3, 4, 7, 5, 6, 0, 8]
[1, 2, 3, 4, 5, 0, 6, 7, 8]
Success
```

3. Implement Iterative deepening search algorithm.

```
Aralyse Iterative Deeping Search Mgorithm Demonstrate how 8 Pozzle
problem could be solved using the algorithm Implement the same
 de iterative- deeping- search (soc, torget);
   depth - limit = 0
     wwwle True:
        swut = depth-limited_ search (sorc target, depth_limit, [])
       9) sesult is not none
   print (" success")
      suturn
     depth-limit +=1
   if depth - limit > 30:
        point (" solution not found within depth limit.")
     setwar
de depth_limitel_search (src, target, depth-limit, visited, States):
     if src = = target :
        print . Stati (src)
     return src
   if depeth - Unit ==0:
      seturn None
    visited - statu append (STO)
      poss mover to do = possible mover (svc, visited statu)
      for movee Inposs_moves_todo:
           if nuve not in visited states:
               print - state (move)
         ought = depth - limited - search (move - farget, dupth - limit-
         visita-states)
         if swell is mothere:
     sutton sunt.
```

```
getorn Wone
del possible movee (states, visitel-states).
       b= state. Index (0)
       d = []
    if 6 notin [011,2]:
       d. append Cu']
    1 6 not in [6,7,8]:
       d. append ('d')
     if 6 mot in [2,518]:
         d-append ('r')
     pros-moves :1+ - con = []
      for ind:
          pos-moves it can append (gen (state is, b))
    actum (more_it-center more_it-cen in pos-mores-
          it - can if move - it . can not in visited- states]
    de gent stade, mis ):
          timp = Steet . copy 1)
          ig m == d:
           tump [613], temp [6] = temp[6], temp [6+3]
        elleif m=='u'
            temp[6-3] temp[6] = temp[6] temp[6+3]
           def print - state (state):
        print ( | " E state COI) { state (17)}
                 & State [2]] \N & State [3] }
                 { state [4] } [ state [6] } In state [6] }
                  3 state Ca) 3 . In " 3
          SIC= [1,2,30,4,51678]
          forget = [1,213,4,510,6,718]
```

0 2 3 1 4 5 6 7 8	123 645 078	123		203 145 678
123645	123	123 405 678	(03 u25 678	123 125 608
123 450 678	123 450 678 success	apl Laugher (de	St.	12021 - 239 12021 - 239 1229 1029 - miles

Implement Iterative deepening search algorithm.

```
def iterative_deepening_search(src, target):
  depth_limit = 0
  while True:
    result = depth_limited_search(src, target, depth_limit, [])
     if result is not None:
       print("Success")
       return
     depth_limit += 1
     if depth_limit > 30: # Set a reasonable depth limit to avoid an infinite loop
       print("Solution not found within depth limit.")
       return
def depth_limited_search(src, target, depth_limit, visited_states):
  if src == target:
    print_state(src)
     return src
  if depth\_limit == 0:
     return None
  visited_states.append(src)
  poss_moves_to_do = possible_moves(src, visited_states)
  for move in poss_moves_to_do:
     if move not in visited_states:
       print_state(move)
       result = depth_limited_search(move, target, depth_limit - 1, visited_states)
```

```
if result is not None:
          return result
  return None
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 [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 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
def print_state(state):
  print(f"\{state[0]\} \{state[1]\} \{state[2]\} \setminus \{state[3]\} \{state[4]\} \{state[5]\} \setminus \{state[6]\} \}
{state[7]} {state[8]}\n")
print("Example 1")
src = [1,2,3,0,4,5,6,7,8]
target = [1,2,3,4,5,0,6,7,8]
print("Source: " , src)
print("Goal State: " , target)
iterative_deepening_search(src, target)
```

OUTPUT

```
Example 1
Source: [1, 2, 3, 0, 4, 5, 6, 7, 8]
Goal State: [1, 2, 3, 4, 5, 0, 6, 7, 8]
0 2 3
1 4 5
6 7 8
1 2 3
6 4 5
078
1 2 3
4 0 5
6 7 8
0 2 3
1 4 5
6 7 8
2 0 3
1 4 5
6 7 8
1 2 3
6 4 5
078
1 2 3
6 4 5
7 0 8
1 2 3
4 0 5
6 7 8
```

```
1 0 3
4 2 5
6 7 8

1 2 3
4 7 5
6 0 8

1 2 3
4 5 0
6 7 8

1 2 3
4 5 0
6 7 8

Success
```

4. Implement A* search algorithm.

```
Report queue as a

qual = [[1/213], [4/5/6], [218/0]]

del is Gual (State):

outurn state = qual

del tensistic Value (State):

(nt = 0

for i in stange (un(qual)):

forj in range (lan(qual)(i]):

if qual [:][j]! = state [:][j]:

cut += 1

outurn cut.
```

```
def get coordinates (werent Hate)
       for 9 in varye (un (goal)):
         bor i in range (len (goal li]))-
              Ef merent State (ijij] = = 0:
                suttorn (i,j)
def is valid (i,j) -> Lool:
      seturn 0 <- 1 < 3 and 0 <= j < 3
     A- stan (state, god) - o not:
       visited = set ()
       pg = a. Powority aveve ()
 while not pq. empty ()
       movel, warent state = pg. get ()
     if usement State = goal:
 if tuple (may (tuple, wowent State))
          " n vished:
   Continue
   visited. add (tuple tmap (tuple, warent state))
  wordinates eget wordinates (werent state)
    1) = wordinates [0], wordinates [1]
  for drasty en[Poil (0-1), (10), (-1,0)]:
   if is valid (new - i, new - j):
        new-state = [row 1: ] for row in wevent state ]
       new-state [i][j], new-state
         (mew-i] [new-j] = new-state [new-i] (new-j],
           new-state (IJJ)
```

```
state = [(1,2,3], [4,0,5], [6,7,8])

mover = A. star (State, goal)

flowver ==-1:

print ("No way to suach the given state")

else:

print ("Readud in" + Str (mover) + "mover")

OUTPUT:

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

Readud in 14 mover,
```

${\bf Implement}\, {\bf A*}\, {\bf search}\,\, {\bf algorithm}.$

```
def print_grid(src):
  state = src.copy()
  state[state.index(-1)] = ' '
  print(
     f"""
{state[0]} {state[1]} {state[2]}
{state[3]} {state[4]} {state[5]}
{state[6]} {state[7]} {state[8]}
  )
def h(state, target):
  #Manhattan distance
  dist = 0
  for i in state:
     d1, d2 = state.index(i), target.index(i)
     x1, y1 = d1 \% 3, d1 // 3
     x2, y2 = d2 \% 3, d2 // 3
     dist += abs(x1-x2) + abs(y1-y2)
  return dist
def astar(src, target):
  states = [src]
  g = 0
  visited_states = set()
  while len(states):
     moves = []
     for state in states:
```

```
visited_states.add(tuple(state))
       print_grid(state)
       if state == target:
          print("Success")
          return
       moves += [move for move in possible_moves(state, visited_states) if move not in
moves]
     costs = [g + h(move, target) for move in moves]
     states = [moves[i] for i in range(len(moves)) if costs[i] == min(costs)]
    g += 1
  print("Fail")
def possible_moves(state, visited_states):
  b = state.index(-1)
  d = []
  if 9 > b - 3 >= 0:
    d += 'u'
  if 9 > b + 3 >= 0:
    d += 'd'
  if b not in [2,5,8]:
     d += 'r'
  if b not in [0,3,6]:
    d += '1'
  pos_moves = []
  for move in d:
     pos_moves.append(gen(state,move,b))
  return [move for move in pos_moves if tuple(move) not in visited_states]
def gen(state, direction, b):
  temp = state.copy()
  if direction == 'u':
```

```
temp[b-3], temp[b] = temp[b], temp[b-3]
  if direction == 'd':
     temp[b+3], temp[b] = temp[b], temp[b+3]
  if direction == 'r':
     temp[b+1], temp[b] = temp[b], temp[b+1]
  if direction == 'l':
     temp[b-1], temp[b] = temp[b], temp[b-1]
  return temp
#Test 1
print("Example 1")
src = [1,2,3,-1,4,5,6,7,8]
target = [1,2,3,4,5,-1,6,7,8]
print("Source: ", src)
print("Goal State: " , target)
astar(src, target)
#Test 2
print("Example 2")
src = [1,2,3,-1,4,5,6,7,8]
target=[1,2,3,6,4,5,-1,7,8]
print("Source: " , src)
print("Goal State: " , target)
astar(src, target)
#Test 3
print("Example 3")
src = [1,2,3,7,4,5,6,-1,8]
```

```
target=[1,2,3,6,4,5,-1,7,8]
print("Source: " , src)
print("Goal State: " , target)
astar(src, target)
```

OUTPUT:

```
Example 1
Source: [1, 2, 3, -1, 4, 5, 6, 7, 8]
Goal State: [1, 2, 3, 4, 5, -1, 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
Success
Example 2
Source: [1, 2, 3, -1, 4, 5, 6, 7, 8]
Goal State: [1, 2, 3, 6, 4, 5, -1, 7, 8]
1 2 3
  4 5
6 7 8
1 2 3
6 4 5
  7 8
Success
```

```
1 2 3
Example 3
Source: [1, 2, 3, 7, 4, 5, 6, -1, 8]
Goal State: [1, 2, 3, 6, 4, 5, -1, 7, 8]
                                                                              6 5
                                                                              4 7 8
1 2 3
7 4 5
                                                                              1 2 3
                                                                             6 5
4 7 8
6 8
1 2 3
                                                                              1 2 3
7 4 5
                                                                              6 7 5
  6 8
                                                                              4 8
1 2 3
                                                                             1 2 3
 4 5
                                                                             6 7 5
4 8
7 6 8
 2 3
                                                                              1 2 3
1 4 5
                                                                              7 5
7 6 8
                                                                              6 4 8
1 2 3
                                                                              2 3
4 5
7 6 8
                                                                              1 7 5
                                                                              6 4 8
1 2 3
                                                                             1 2 3
4 6 5
                                                                              7 5
6 4 8
```

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

5. Implement vacuum cleaner agent.

```
Vacuom Cleaner Agent
del vacuum - world ():
      goal _ state = { A: O; B: 0}
      location Input = "input (" Enter location of Vaccum")
      Status input input ("Enter status of "+" location. input)
      Statul_input - complement = input ("Enter Statule of other
     print ("Initial location = input (" unter" landition" + Str
              (goal -state)
     "b location_"input = = A':
        print ("Vaccum is placed in location A")
         of statul imput == 11: 1 = 1000 mg
         print ("Location A is Dirty: ")
          goal = State [71] = 0
           wat+ = 1
     print (" LOST GOY CLEAMING A" + Str (LOST))
     print ("location A has been cleaned)
  1/2 Status - input - complement = = "1":
       print (" location B " Divty ")
        print (" Howing sught to the loce B")
          Wh+ += 1
       print (" lost for moving RIGHT" + IT (LOST))
          goal-state (B'] = 0
          was +=1
         print (11 LOST for SUCK 4 +Str (LOSH))
         print (" LO (B has been cleaned.")
   elle:
       Print ("No action" + Str (08+))
       print (" LOCBis already clean.")
```

```
Il Status _ input = = 0:
    print ("LOCA is already clean.")
    1) Status - input - tomplement == 1!
        Print ("LOCB is Dirty !" )
     print (" Housing RIGHT to the LOCB".)
       ws++=1
    print (" wet for suck" + styllost))
    print ( " LOC 8 has been cleaned. ")
  else:
      print ("No action" F Str ((OB+))
      print (lost)
      privat ("LOC 8" to already clean").
  print (" cost for suck" + str(cox+)
  print ("LOCA has been cleaned")
  elee:
     Print (LOSH)
      print (" LOC B is already clean: ")
      1) Status-input - complement = = '1':
          print ("LOCA is Dirty ")
          print (" Howing LEFT to the LOCA")
        print (" LOST for SUCK"+Strllor+))
         print ("Loc A har Leen (leaned. ")
    else:
       print (" No action " + str (count))
       print ("LOCA is already clean:")
 print ( " GOAL STATE : ")
  print (goal - State)
  print ("performing measurement"- (str (cost))
```

Enter status of other room O

Intial (ocation (condition { N'; O', B'; O'})

Vacuum ?s placed & location B

LOCATION B is Dirty

COST for waning 1

(Ocation B has been closhed

GNAL STATE:

{ N'; b', B'; O'}

Implement vacuum cleaner agent.

```
def clean(floor, row, col):
  i, j, m, n = row, col, len(floor), len(floor[0])
  goRight = goDown = True
  cleaned = [not any(f) for f in floor]
  while not all(cleaned):
     while any(floor[i]):
       print_floor(floor, i, j)
       if floor[i][j]:
          floor[i][j] = 0
          print_floor(floor, i, j)
       if not any(floor[i]):
          cleaned[i] = True
          break
       if j == n - 1:
          i -= 1
          goRight = False
       elif j == 0:
          j += 1
          goRight = True
       else:
          j += 1 if goRight else -1
     if all(cleaned):
       break
     if i == m - 1:
       i = 1
       goDown = False
     elif i == 0:
       i += 1
```

```
goDown = True
     else:
       i += 1 if goDown else -1
     if cleaned[i]:
       print_floor(floor, i, j)
def print_floor(floor, row, col): # row, col represent the current vacuum cleaner position
  for r in range(len(floor)):
     for c in range(len(floor[r])):
       if r == row and c == col:
          print(f" >{floor[r][c]}< ", end = ")
       else:
          print(f" {floor[r][c]} ", end = ")
     print(end = '\n')
  print(end = '\n')
#Test 1
floor = [[1, 0, 0, 0],
     [0, 1, 0, 1],
     [1, 0, 1, 1]]
print("Room Condition: ")
for row in floor:
  print(row)
print("\n")
clean(floor, 1, 2)
```

```
Room Condition:
                                                                 0
                                                      1
                                                            0
                                                                      0
[1, 0, 0, 0]
                                                       0
                                                            0
                                                                 0
                                                                      0
[0, 1, 0, 1]
                                                      >1<
                                                            0
                                                                 1
                                                                      1
[1, 0, 1, 1]
                                                      1
                                                            0
                                                                 0
                                                                      0
                                                      0
                                                            0
                                                                 0
                                                                      0
                                                      >0<
                                                            0
                                                                 1
 1
       0
            0
                   0
       1
                   1
  0
            >0<
                                                      1
                                                            0
                                                                 0
                                                                      0
  1
       0
             1
                   1
                                                       0
                                                            0
                                                                 0
                                                                      0
                                                       0
                                                           >0<
  1
       0
             0
                  0
  0
       1
             0
                 >1<
                                                            0
                                                                 0
                                                                      0
  1
       0
             1
                  1
                                                       0
                                                            0
                                                                 0
                                                                      0
                                                       0
                                                            0
                                                                >1<
  1
       0
             0
                  0
  0
       1
             0
                  >0<
                                                       1
                                                            0
                                                                      0
                                                                 0
                                                       0
                                                            0
                                                                 0
                                                                      0
  1
       0
             1
                   1
                                                       0
                                                            0
                                                                >0<
                                                                      1
  1
             0
       0
                   0
                                                       1
                                                            0
                                                                 0
                                                                      0
  0
       1
            >0<
                   0
                                                       0
                                                            0
                                                                 0
                                                                      0
       0
  1
             1
                   1
                                                       0
                                                            0
                                                                 0
                                                                     >1<
  1
       0
             0
                   0
                                                       1
                                                            0
                                                                 0
                                                                      0
      >1<
             0
                   0
  0
                                                       0
                                                            0
                                                                 0
                                                                      0
  1
       0
             1
                   1
                                                       0
                                                            0
                                                                 0
                                                                     >0<
  1
       0
             0
                   0
                                                       1
                                                            0
                                                                 0
                                                                      0
  0
      >0<
             0
                   0
                                                       0
                                                            0
                                                                 0
                                                                     >0<
  1
       0
             1
                   1
                                                       0
                                                            0
                                                                 0
                                                                      0
                                                                     >0<
                                                       1
                                                            0
                                                                 0
  1
       0
             0
                   0
                                                       0
                                                            0
                                                                 0
                                                                      0
                   0
  0
       0
             0
                                                                 0
                                                                      0
                                                       0
                                                            0
  1
      >0<
             1
                   1
```

```
1
       0
            >0<
                   0
 0
       0
             0
                   0
 0
      0
             0
                   0
 1
      >0<
             0
                   0
0
      0
             0
                   0
 0
       0
             0
                   0
>1<
      0
             0
                   0
0
       0
             0
                   0
0
       0
             0
                   0
>0<
             0
                   0
       0
0
       0
             0
                   0
 0
       0
             0
                   0
```

6. Create a knowledge base using prepositional logic and show that the given query entails the knowledge base or not.

Create KB using prepositional logic and ST tu given def-exaluate-exp (9,p, x): exp. secult = (Porg,) and (not vorp) suctorn exp_swell . de generate - HU: print ("It &p (KB)") for 9 in [True, False]: for p in ETWE, False J: porr in liver Fall 1: at long to be sook person exp. sult = evaluate_exp (9,9,x) query - squt = paralr.

```
def query-entails - Knauloge ().
    for q in three Fall!
    for p in [" ", Falle]: Novaled 908 ] -
     exp syll = evaluate _exp (q,p,r)
      grong-runt = p &r
      statum falle.
 setum True
  dy main()
    garrate - truth-table ()
       print ("In away entails with traveledge"):
   de
      √-name = = "- main-":
main()
  Owtput:
                      def- evaluate - exp (apr 1)
  Expression (KB)
   True True
   True | Falle
   Falle | Falle
   True I Falle
   The True
   True | False
   Fall Fall.
   Coverey does not entail the Knowledge
```

Create a knowledge base using prepositional logic and show that the given query entails the knowledge base or not.

```
def evaluate_expression(p, q, r):
  expression_result = (p \text{ or } q) and (not r \text{ or } p)
  return expression_result
def generate_truth_table():
  print(" p | q | r | Expression (KB) | Query (p^r)")
  print(" | | | ")
  for p in [True, False]:
     for q in [True, False]:
       for r in [True, False]:
          expression_result = evaluate_expression(p, q, r)
          query_result = p and r
          print(f" \{p\} \mid \{q\} \mid \{r\} \mid \{expression\_result\} \qquad \qquad | \{query\_result\}")
def query_entails_knowledge():
  for p in [True, False]:
     for q in [True, False]:
       for r in [True, False]:
          expression_result = evaluate_expression(p, q, r)
          query_result = p and r
          if expression_result and not query_result:
             return False
  return True
```

```
def main():
    generate_truth_table()

if query_entails_knowledge():
    print("\nQuery entails the knowledge.")

else:
    print("\nQuery does not entail the knowledge.")

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

```
KB: (p or q) and (not r or p)
            Expression (KB)
                              Query (p^r)
            | True | True
                                          True
             False | True
                                           False
               True | True
 True | False
               False | True
                                            False
 False | True
               True | False
 False
       True | False | True
 False | False | True | False
                                              False
 False | False | False
Query does not entail the knowledge.
```

7. Create a knowledge base using prepositional logic and prove the given query using resolution

Create a knowledge base using prepositional lage and prove the import re de main (sules, goal): oules = ouler split ('') Steps = gusolive (suiler, goal) print (" In step 1 + 1 clawelt | Derivation It') print (1-1*30) dy negate (term): seturn b'a stang' of tom FODI = 'a' else termill. def severse (dame):

if len (clame)>2:

t= split-terme (clame)

```
sutorn 1
dy split - terms (seule) = [ ] along 3 1 ] = part
 eptit terms ('~ pur)
  ['NP', R']
 /* dy split - terms (sude):
     exp= (N*[Fars]) op 1 Loop ) As its books of
  terms = re-findall [exp, rule]
     suturn termy */
  def sudve (suler, goal):
    temp = sules. copy 1) has such as the summer
   temp + = [megati (goal)]
    steps = dict()
    for rule in tup: (mall lugge got
        Steps Caule] . Gruen' AN (1+1) =1
    Steps [night God)] = Negated Conclusion!
      1=0
      volute iclen (tomp): All 909 and 909 and
         n = (on Ctemp)
       J= (1+1):1.n
          clanes = []
          wile 1 = 1:
            term 1 - spot time
            term 2 = split -terms
            for cinterny 1:
            gen = £1++2
             il (en(gen) == 2:
```

```
gen [0]! = regati (gen [i]).
     clauses += [1' [ gen [0] ]
        vigen (173)
elle:
   il contradiction (goal, 1' & gen [0]}, Vigen []]:
      V Egen [1]]') James que ] makent ex = somet
  Steps ["] = f" (Resolved)
      clause on clauses:
       il clause not intemp and clause v = 1000
          sierense (clause) and sieverse (clause)
          not the temp
          tump. append (Maure)
    j= (j+1) 1.n
 1+=1
 suturn Steps
sules = RVAP RVAD NRVA NRVA
  goal = R'
 main (outer, goal)
  900d = R'
 main (sules, goal)
  moin (outer, 12)
```

```
output :-
                    Degluation
           Uame 1
     Step
                      Gruen Land 1 1
           1 RUP
            Rund of Grown Mandard - House are
                      Given ? 9 - Harry provis
             NRUP
            NRVE
                      biven
                     pregeted conclusion
      5
             NR
                      Ruolved Rung and MRUPTO RUNG,
    which is well
    A lastradiction Esfound when MR is assumed as true.
      HERUTER THE DAY HOW STOOTHS HAVE NOW THIS
```

Create a knowledge base using prepositional logic and prove the given query using resolution

```
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' \sim \{\text{term}\}' \text{ if } \text{term}[0] != '\sim' \text{ else } \text{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 = '(\sim *[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 \text{ for } t \text{ in terms } 1 \text{ if } t != c]
             t2 = [t \text{ for } t \text{ in terms } 2 \text{ 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 \sim P Rv \sim Q \sim RvP \sim RvQ' \#(P^{\wedge}Q) <=>R : (Rv \sim P)v(Rv \sim Q)^{\wedge}(\sim RvP)^{\wedge}(\sim RvQ)
goal = 'R'
print('Rules: ',rules)
print("Goal: ",goal)
main(rules, goal)
rules = PvQ \sim PvR \sim QvR' \#P=vQ, P=>Q : \sim PvQ, Q=>R, \sim QvR
goal = 'R'
print('Rules: ',rules)
print("Goal: ",goal)
main(rules, goal)
rules = 'PvQ PvR \simPvR RvS Rv\simQ \simSv\simQ' # (P=>Q)=>Q, (P=>P)=>R, (R=>S)=>\sim(S=>Q)
                   'R'
goal
          =
print('Rules: ',rules)
```

```
print("Goal: ",goal)
main(rules, goal)
```

```
Example 1
Rules: Rv~P Rv~Q ~RvP ~RvQ
Goal: R
        |Clause |Derivation
Step
 1.
          Rv~P
                  Given.
                  Given.
          Rv~Q
 2.
 3.
         ~RvP
                  Given.
 4.
          ~RvQ
                  Given.
5.
          ~R
                  Negated conclusion.
                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.
Example 2
Rules: PvQ ~PvR ~QvR
Goal: R
        |Clause |Derivation
Step
          Pv0
                 Given.
 1.
          ~PvR
                  Given.
 2.
                  Given.
          ~QvR
 3.
 4.
          ~R
                  Negated conclusion.
          QvR
                Resolved from PvQ and ~PvR.
 5.
                Resolved from PvQ and ~QvR.
 6.
          PvR
         ~P
                Resolved from ~PvR and ~R.
 7.
                 Resolved from ~QvR and ~R.
 8.
          ~Q
                  Resolved from ~R and QvR.
 9.
          Q
          Р
 10.
                  Resolved from ~R and PvR.
          R
 11.
                 Resolved from QvR and ~Q.
                 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.
```

```
Rules: PvQ PvR ~PvR RvS Rv~Q ~Sv~Q
Goal: R

Step | Clause | Derivation

1. | PvQ | Given.
2. | PvR | Given.
3. | ~PvR | Given.
4. | RvS | Given.
5. | Rv~Q | Given.
6. | ~Sv~Q | Given.
7. | ~R | Negated conclusion.
8. | QvR | Resolved from PvQ and ~PvR.
9. | Pv~S | Resolved from PvQ and ~Sv~Q.
10. | P | Resolved from PvR and ~R.
11. | ~P | Resolved from ~PvR and ~R.
12. | Rv~S | Resolved from ~PvR and ~R.
13. | R | Resolved from ~PvR and ~R.
14. | S | Resolved from RvS and ~R.
15. | ~Q | Resolved from RvS and ~R.
16. | Q | Resolved from Rv~S and ~R.
17. | ~S | Resolved from ~R and Rv~S.
18. | Resolved from ~R and Rv~S.
18. | Resolved from ~R and Rv~S.
19. | Resolved from ~R and Rv~S.
20. | Resolved from ~R and Rv~S.
21. | Resolved from ~R and Rv~S.
22. | Resolved from ~R and Rv~S.
23. | Resolved from ~R and Rv~S.
24. | Resolved from ~R and Rv~S.
25. | Resolved from ~R and Rv~S.
26. | Resolved from ~R and Rv~S.
27. | Resolved from ~R and Rv~S.
28. | Resolved from ~R and Rv~S.
29. | Resolved from And Rv~S.
20. | Resolved from Rv~S.
20. |
```

8. Implement unification in first order logic

```
First order
def villy (expr1, expr2)
  # split expressions into functions organists funct 1, org = 2 = expr 1, split ('c',1)
    Junit 2, angs 2 = expr 2. split ( c',1)
if funcs! = func 21
 print ( "expressione cannot be unfied Different functione")
   Justom Mone
agss = agss - vetip ('1') split (',')
ongs 2 = angs 2 . rstrip ("1"). split (",")
 substitution = {}
for Q1, Q2 in zip (args1, args2):
       if as. is sower () and az. islower() and
            all = a2:
        substitution [al] = a2
       dif al. is tower () and not az is lower ():
           Substitution [a1] = a2
       et not a 1 is lower () and a2 is hower ();
               substitution [az]=91
        dh 011 - 02;
            print ("Expression cannot be unfed
                   Incompatible congument (!);
               autorn None
```

outurn substitution dy apply - substitution (exp. , substitution) forky, value in substitution, items (); expr = expr suplace (ky, value) outern expr Output :- enter 1st expr: stn(n) " 2nd " " : cos(a) Expressions cannot be wifted tillevent furtions Output: Ender 1st expr: add (m,y) n/a - substitutions 410 2 - substitutione add (0.6) add (C.6)

Implement unification in first order logic

```
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 is Variable (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 is Variable (exp1):
  if checkOccurs(exp1, exp2):
     return False
  else:
     return [(exp2, exp1)]
if is Variable (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)
  remaining Substitution = unify (tail1, tail2) \\
  if not remainingSubstitution:
     return False
  initialSubstitution.extend(remainingSubstitution)
  return initialSubstitution
print("\nExample 1")
\exp 1 = \text{"knows}(f(x),y)\text{"}
exp2 = "knows(J,John)"
print("Expression 1: ",exp1)
print("Expression 2: ",exp2)
substitutions = unify(exp1, exp2)
print("Substitutions:")
print(substitutions)
print("\nExample 2")
exp1 = "knows(John,x)"
```

```
exp2 = "knows(y,mother(y))"
print("Expression 1: ",exp1)
print("Expression 2: ",exp2)

substitutions = unify(exp1, exp2)
print("Substitutions:")
print(substitutions)

print("\nExample 3")
exp1 = "Student(x)"
exp2 = "Teacher(Rose)"
print("Expression 1: ",exp1)
print("Expression 2: ",exp2)

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

```
Example 1

Expression 1: knows(f(x),y)

Expression 2: knows(J,John)

Substitutions:

[('J', 'f(x)'), ('John', 'y')]

Example 2

Expression 1: knows(John,x)

Expression 2: knows(y,mother(y))

Substitutions:

[('John', 'y'), ('mother(y)', 'x')]

Example 3

Expression 1: Student(x)

Expression 2: Teacher(Rose)

Predicates do not match. Cannot be unified

Substitutions:
False
```

9. Convert a given first order logic statement into Conjunctive Normal Form (CNF).

```
Convert FOL
                   Statement to CNF
del get Attributes (Strag):

expr = 1 ([n)] + ()
       matches = sue. findall (expr. string)
       seturne Emfor min str (matches ) if M. isalpha () ]
dy explemization (stadement):
      SKOLEM_CONSTANTS = [ ] & UN (C)3 for
      metches: outfindall ('[7'., Statement).
          for match in matches [::-1]:
             Statement : Statement . suplace (match)
             for predicate in get predicates (Statement):
            attributel - get Atributes (predicate)
    suction stadement
        Emport Xo.
 dy for to - ung (for):
     Statement = for suplace (">", "=")

expr = "(C(C^J]+)1]'
        Statements - see find all (expr. statement)
          for i, s in enumorate (statements)

if 'E' in s and J' not in s!
           statements Pi] += ']'
          101 s in Statements
       Statement - Statement (: 5,7+news --
```

print (for- to- (ry) ("] or (findless) => ~ (ry (2))" Output: Capird (as Infly (x) Economics in the meson to stand on the ekulumisten (stalumit)) SKOLEM, CONSTRAINS = [] & LANCON BON The : I washen is waters +6) Rollings . September . topical (motos) for predicate in get Predictor Contents I attacked - get the suffer of presidents ely for to any (fold: Statement = for supplied (25 = 1)

Convert a given first order logic statement into Conjunctive Normal Form (CNF).

```
def getAttributes(string):
  expr = ' ([^{\wedge})] + )'
  matches = re.findall(expr, string)
  return [m for m in str(matches) if m.isalpha()]
def getPredicates(string):
  expr = '[a-z\sim]+\backslash([A-Za-z,]+\backslash)'
  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)]"))

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)"))
```

```
Example 1
FOL: bird(x)=>~fly(x)
CNF: ~bird(x)|~fly(x)

Example 2
FOL: ∃x[bird(x)=>~fly(x)]
CNF: [~bird(A)|~fly(A)]

Example 3
FOL: animal(y)<=>loves(x,y)
CNF: ~animal(y)<|loves(x,y)

Example 4
FOL: ∀x[∀y[animal(y)=>loves(x,y)]]=>[∃z[loves(z,x)]]
CNF: ∀x~[∀y[~animal(y)|loves(x,y)]]|[[loves(A,x)]]

Example 5
FOL: [american(x)&weapon(y)&sells(x,y,z)&hostile(z)]=>criminal(x)
CNF: ~[american(x)&weapon(y)&sells(x,y,z)&hostile(z)]|criminal(x)
```

10. Create a knowledge base consisting of first order logic statements and prove the given query using forward reasoning.

```
Clar Employees
import re
del is noglaph (2):
     go usefable (x):
sustain lenta) == 1 and x. Polower () and x. To alpha ()
                            resultante - moderne des
day get Historie (String): (2) top monthly =)
nations = ac findal corps, cting) [[] tool - we too
  suturn matches
                                  ( Ua) per ) italians deb
dy got prediate (sting)

E7 = 2000 total

expr = '([a-z-J-b]([-1]]+1)'

E7 = 2000 total

expr = '([a-z-J-b]([-1]]+1)'
   seutorn de findell (expr. strig) 1200 07 00 14
                         : With so hi by rej .
 closs
      Fatt:
 ely - Prit (seed expression):
  Self expression = supression
  self prediate : prediate self content ())
   predicale get paradicales (expression) [b]

passong - get Attributes
   def sustitute (set, wortends): dustrial of personal of
     is variables (6) elle Plur pin sul.
```

selturn Fact () Clos Emplication: 1007 3 George of pet dub - Port - (sept expression): ceet expression - expression of her marked (start). 1 = expression = split ('>') Seet the = [Fail (b) fort in tho] spect to] southern methods det evoluate (sup, jat): (mits) whothere top the Contents = 83 new-lhs = C) (1+[10.1]/4[-5-0] 3" = 17" for fout in facts: (part igner) blooding metus for val in sect-ths; if val-prodicate = - fait predicate for: , v in crumerate (val. get varianco): ib v: Constant [v] = fact get consent of [1] new_shs. oppend 16045. (mill form the) more than the predicate attribute - get predicates (sect. The expression b) SAY (Set Attributes (Seed, TN. +1) solutioned top proces? for key in contends "(southern , 600) - studies los il without (Kay): (1940 - and) attributel: attributes explane (key, continue (key)). cxpr = 1 2 Posedicase 22 attributes &

```
doll KB:
   def -- "int -- (seets:
       Selt · facts - Set ()
        Selt. Implication - sett)
    dy tell (section):
      1 6 m (:
      selt. Implication:
     su = : walvite ( seet , jacks)
      if sop!
        (ene) popo · opo (an)
   Kb - = KBD
    Kb - tell ( ling ( M) greedy ( M) = ) evil ( M)
   Kb - tell ( 'King ( Fichead ) ')
    Kb-query (resul (207)
```

Create a knowledge base consisting of first order logic statements and prove the given query using forward reasoning.

```
import re
def isVariable(x):
  return len(x) == 1 and x.islower() and x.isalpha()
def getAttributes(string):
  expr = ' ([^{\wedge})] + )'
  matches = re.findall(expr, string)
  return matches
def getPredicates(string):
  expr = '([a-z\sim]+)\backslash([^{\&}]+\backslash)'
  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'\setminus 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'\setminus 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()
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)')
```

```
Example 1
Querying criminal(x):
        1. criminal(West)
All facts:
        1. american(West)
        2. enemy(Nono,America)
        3. hostile(Nono)
        4. sells(West,M1,Nono)
        5. owns(Nono,M1)
        6. missile(M1)
        7. weapon(M1)
        8. criminal(West)
Example 2
Querying evil(x):
        1. evil(John)
```