VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

ARTIFICIAL INTELLIGENCE

Submitted by

HARIKA N (1BM21CS071)

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
Oct 2023-Feb 2024

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 **"COMPILER DESIGN"** carried out by **HARIKA N (1BM21CS071)**, 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 year 2022-23. The Lab report has been approved as it satisfies the academic requirements in respect of Artificial Intelligence Lab - **(22CS5PCAIN)**work prescribed for the said degree.

Swathi Sridharan
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

INDEX

Name Harika N

Standard Section Roll No.

Subject

Sl.No.	Date	Title	Page No	Teacher's Sign
	10/11/23	Python Programs		
1	17/11/23	Tic Tac Toe game		
2	24/11/23	8-puzzle using BFS		
3	8/12/23	8-puzzle using IDDFS		
4.	8/12/23	8-puzzle using A*		
5.	22/12/23	Vacuum Cleaner agent		
6.	29/12/23	Knowledge Base entailment		
7.	29/12/23	Knowledge Base Resolution		
8.	12/1/24	Unification		
9.	12/1/24	FOL to CNF		
10	12/1/24	Forward Reasoning		

14/11/23

Date :
Page No :

Lab Program 1 - Tic Tac Toe Game

~~bo~~

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

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

def isWinner(trc, pos):
    if trc[0]==trc[4] and trc[4]==trc[8] or
        trc[2]==trc[4] and trc[4]==trc[6]:
    return True
    elif trc[pos-0]==trc[pos-3] and trc[pos-3]==trc[pos-
        return True
    elif trc[pos||3+1]=trc[pos||3+2]==trc[pos||3+3]:
        return True
    return False

def update-user(trc):
    num = int(input("Enter a number on the board"))
    while(num not in trc):
        num = int(input("Enter a number on the
                        board"))
    trc[num-1]=='0'
```

```
def update_comp(trc):
    for i in trc:
        if 'i' != 'x' and 'i' != 'o':
            trc[i-1] = 'x'
            if (isWinner(trc, i-1) == True):
                return
            else:
                trc[i-1] = i

    for i in trc:
        if i != 'x' and i != 'o':
            trc[i-1] = 'o'
            if (isWinner(trc, i-1) == False):
                return
            else
                trc[i-1] = i

num = random.rand(9)
while (num not in trc):
        num = random.rand(9)
trc[num-1] = 'x'
```

Date :
Page No :

Algorithm

Step 1: Make a board & initialize the values
2: Make a winner function, check for the winning possibilities
3: Make a user function which will take no. as a input
4: Make a computer function
   i) Maximize computer winning
   ii) Minimize user winning
   iii) Generate random no and initialize the computer no
5: In the main function ~~st~~ call the user & computer function alternatively.
   After 5 moves check for the winner.
6: print the winner.

OUTPUT

```
Harika N (1BM21CS071)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
+-------------------------------+
|       |       |       |
|   1   |   2   |   3   |
|       |       |       |
+-------------------------------+
|       |       |       |
|   4   |   5   |   6   |
|       |       |       |
+-------------------------------+
|       |       |       |
|   7   |   8   |   9   |
|       |       |       |
+-------------------------------+
computer's turn :
+-------------------------------+
|       |       |       |
|   1   |   X   |   3   |
|       |       |       |
+-------------------------------+
|       |       |       |
|   4   |   5   |   6   |
|       |       |       |
+-------------------------------+
|       |       |       |
|   7   |   8   |   9   |
|       |       |       |
+-------------------------------+
Your turn :
enter a number on the board :4
```

```
Your turn :
enter a number on the board :4
+-------------------------------+
|       |       |       |
|   1   |   X   |   3   |
|       |       |       |
+-------------------------------+
|       |       |       |
|   0   |   5   |   6   |
|       |       |       |
+-------------------------------+
|       |       |       |
|   7   |   8   |   9   |
|       |       |       |
+-------------------------------+
computer's turn :
+-------------------------------+
|       |       |       |
|   X   |   X   |   3   |
|       |       |       |
+-------------------------------+
|       |       |       |
|   0   |   5   |   6   |
|       |       |       |
+-------------------------------+
|       |       |       |
|   7   |   8   |   9   |
|       |       |       |
+-------------------------------+
Your turn :
enter a number on the board :5
+-------------------------------+
```

```
+-----------------------------+
Your turn :
enter a number on the board :5
+-----------------------------+
|             |             |             |
|      X      |      X      |      3      |
|             |             |             |
+-----------------------------+
|             |             |             |
|      O      |      O      |      6      |
|             |             |             |
+-----------------------------+
|             |             |             |
|      7      |      8      |      9      |
|             |             |             |
+-----------------------------+
computer's turn :
+-----------------------------+
|             |             |             |
|      X      |      X      |      X      |
|             |             |             |
+-----------------------------+
|             |             |             |
|      O      |      O      |      6      |
|             |             |             |
+-----------------------------+
|             |             |             |
|      7      |      8      |      9      |
|             |             |             |
+-----------------------------+
winner is  X
```

04/11/23

Date :
Page No :

8 puzzle problem

Algorithm

1) Function bfs(src, target)

> Initialize an empty queue and add the source state 'src'.

```
queue = []
queue.append(src)
```

> While the queue is not empty:

```
source ← queue.pop(0)
```

> Append 'source' to the 'exp' list

```
exp.append (source)
```

> If 'source' is equal to target state print "success' and return.

> Generate possible moves from 'source' using possible moves function.

> For each possible move:

If move is not in explored states or queue, add it to the queue.

2) Function possible_moves (state, visited states)

```
b = state.index(0)   ⇐ to find index of empty spot
```

> Initialize directions array

```
d = []
```

> if b not in [0, 1, 2]:

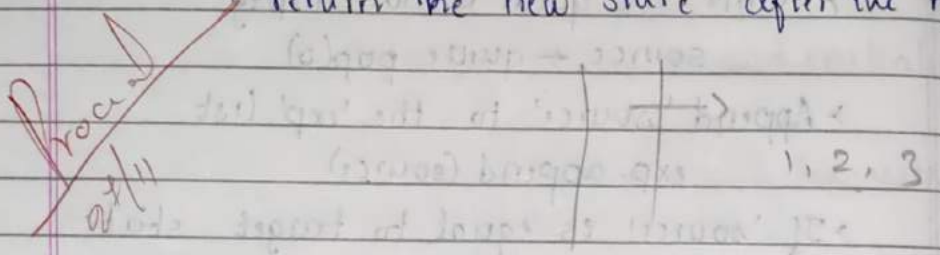
```
d.append('u')
```

> Same way determine possible ~~stat~~ directions to move based on position of empty spot.

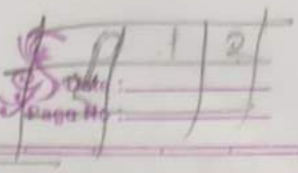
- Generate possible moves in those directions.
- Return a list of possible moves that have not been visited.

3) Function 'gen(state, m, b)'

- Copy the current state to a temporary variable
- perform the move (Swap) based on the direction specified.
- return the new state after the move



source (1, 3, 2, 4, 5, 7, 6, 8, 0)
target (1, 2, 3, 4, 0)



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

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):
    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')
```

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

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

        poss_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_state]

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 temp
```

src = [1,2,3,0,4,5,6,7,8]
target = [1,2,3,4,5,0,6,7,8]
bfs(src, target)

[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

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
4 | 0 | 6
7 | 5 | 8
- - - - - -

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

success

OUTPUT

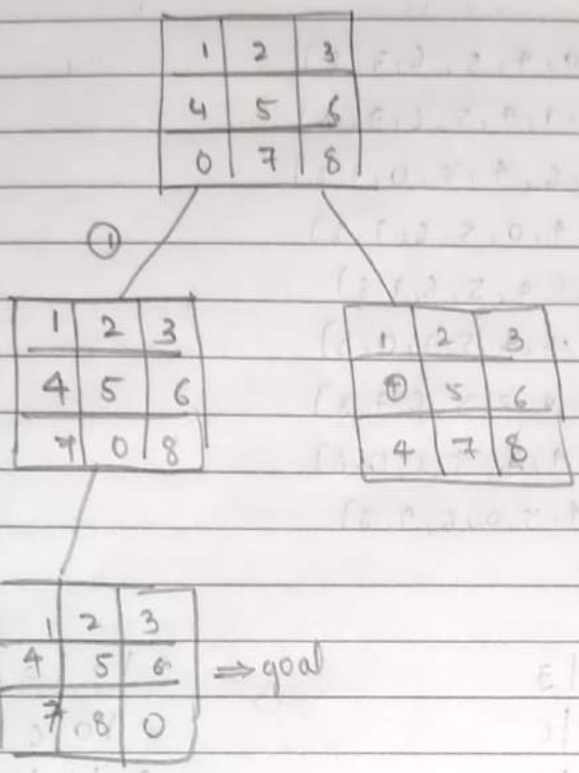
```
Harika N (1BM21CS071)
1 | 2 | 3
4 | 5 | 6
0 | 7 | 8
-----------
1 | 2 | 3
0 | 5 | 6
4 | 7 | 8
-----------
1 | 2 | 3
4 | 5 | 6
7 | 0 | 8
-----------
0 | 2 | 3
1 | 5 | 6
4 | 7 | 8
-----------
1 | 2 | 3
5 | 0 | 6
4 | 7 | 8
-----------
1 | 2 | 3
4 | 0 | 6
7 | 5 | 8
-----------
1 | 2 | 3
4 | 5 | 6
7 | 8 | 0
-----------
Success
```

8/12/23

Date :
Page No :

→ 8 puzzle - iterative deepening search algorithm

Algorithm



Algorithm:-

1) Initialize the initial state = [ ] and goal state for the 8 puzzle.

2) Set the depth = 1 and expand the initial state.

The depth_limited_search (depth) is performed

```
if node.state = goal
        return node
else
        for neighbour in get_neighbour(node.state)
            child = puzzle node(neighbour, node)
            result = depth_limited_search(depth-1)
            if result == True
                return result
```

3) After one iteration where depth=1, increment the depth by 1 and perform depth_limited_search again.

4) Here get_neighbours will generate the possible moves by swapping the '0' tile.

5) The path traversed is printed to reach the goal state.

8/12

OUTPUT

```
Harika N (1BM21CS071)
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]]
```

8/12/23

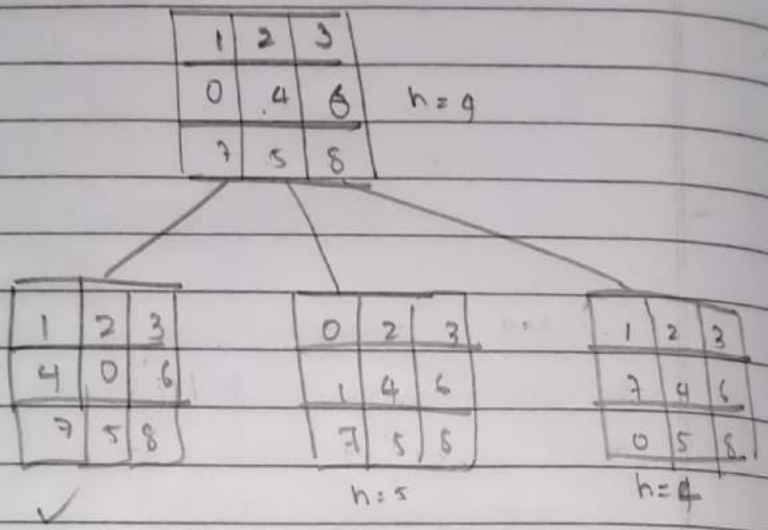
Date :
Page No :

8-puzzle using A* algorithm

goal

1	2	3
4	5	6
7	8	0

h=0 (heuristics, number of misplaced tiles)



Algorithm

1) Create the initial state and goal state for the problem. In A* the heuristics is considered, lower heuristic node is considered in each state.

fvalue = hvalue + pathcost

2) Initially expand the node, find the location of empty tile and generate the nodes. Calculate the heuristic function value

$f(x) = h(x) + g(x)$

$h(x)$ → misplaced tiles

$g(x)$ → depth from starting node (path cost)

3) Maintain two lists namely 'open' and 'close' The nodes (states) generated are stored in the open list, sort using the $f(x)$ values. The

explored nodes are stored in close list and removed from open.

4) The goal is reached when h(x)=0, implies that all the tiles are in the correct position.

8/12

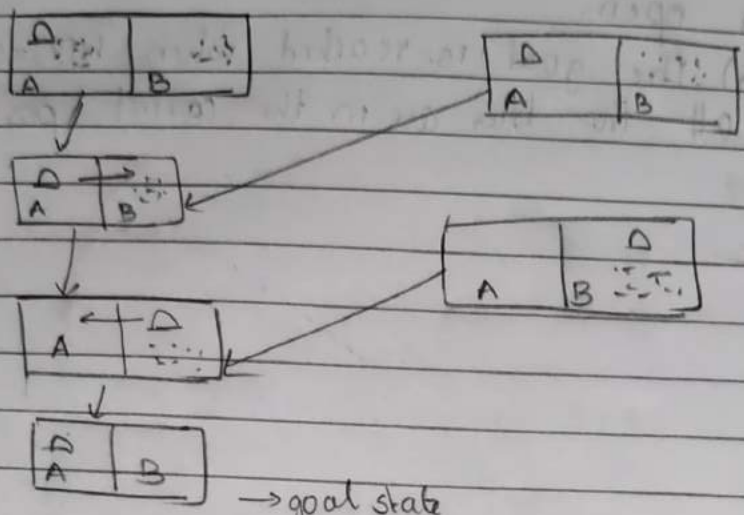
OUTPUT

```
Harika N (1BM21CS071)
Enter the start state matrix

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

1 2 3
4 5 6
7 8 _




  |
  |
 \'/

1 2 3
4 5 6
_ 7 8

  |
  |
 \'/

1 2 3
4 5 6
7 _ 8

  |


  _

  |
  |
 \'/

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

20/12/23

Date :
Page No :

→ Vaccum cleaner agent



→ goal state

Algorithm

1) Initialize the starting & goal state, the goal is to clean both rooms A and B ~~and~~

2) If status = Dirty then clean

else if location = A and status = clean then
return right

else if location = B and status = clean then
return left

else

exit

3) If both the locations are clean the vaccum cleaner is done with its task.

Date :
Page No :

```
def vaccum_world():
    goal_state = {'A': '0', 'B': '0'}
    cost = 0

    location_input = input("Enter location of Vacum")
    status_input = input("Enter status of"
                         + location_input)
    status_input complemun


class VaccumCleaner:
    def __init__(self, initial_location):
        self.location = initial_location

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

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

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

    def simulate_cleaning():
        initial_vaccum_location = input("Enter initial
                location of cleaner(A/B): ").upper()
        vaccum = VaccumCleaner(initial_vaccum_location)
```

Date :
Page No :

```
room_A_state = input("Enter state for Room A (clean/dirty):
                ").lower()
room_B_state = input("Enter state for Room B (clean/
                dirty): ").lower()

rooms = {
    'A' : room_A_state,
    'B' : room_B_state
}

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

if rooms['A'] == 'clean' and rooms["B"] == 'clean':
    print("Both rooms are already clean.
          No cleaning needed.")
else:
    print("Starting the cleaning process...")

    current_room = vacuum.location
    cleaned_room = vacuum.suck(current_room)

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

    #move to other room
    if current_room == 'A':
        vacuum.room_right()
        current_room = 'B'
    else:
```

```
        vacuum.move_left()
        current_room = 'A'

        cleaned_room = vacuum.suck(current_room)
        if cleaned_room == 'clean':
            rooms[current_room] = 'clean'

        print("Cleaning completed.")
        print(f"Final state:")
        print(f"Vacuum Cleaner is in Room {vacuum.loca
                                        -tion}")

        print(f"Room A: {rooms['A']}")
        print(f"Room B: {rooms['B']}")

    simulate_cleaning()
```

Output

```
Enter initial location of vacuum cleaner (A/B): A
Enter state for Room A (clean/dirty): dirty
Enter state for Room B (clean/dirty): dirty

Initial state:
Vacuum Cleaner is in Room A
Room A: dirty
Room B: dirty

Starting the cleaning process...
    sucking dirt in Room A
    Moving right
    sucking dirt in Room B
```

Cleaning completed.
Final state:
Vacuum Cleaner is in Room B
Room A: clean
Room B: clean.

OUTPUT:

```
Harika N (1BM21CS071)
0 indicates clean and 1 indicates dirty
Enter Location of Vacuumb
Enter status of b1
Enter status of other room1
Vacuum is placed in location B
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
```

Date :
Page No :

Knowledge based entailment

Inputs:
Knowledge base (set of logical rules)
Query statement

Steps:
1) Negate the Query
2) Combine with the Knowledge Base using conjunction
3) Check Satisfiability:
- Use a satisfiability check (eg. 'satisfiable') function to determine if the conjunction is satisfiable.
- If conjunction not satisfiable, it means there's no assignment of truth values that satisfies both the knowledge base & the negation of query
4) Determine Entailment
- If conjunction is satisfiable,
return true
else
return false.

Proceed

OUTPUT:

```
Harika N (1BM21CS071)
Knowledge Base: ~r & (Implies(p, q)) & (Implies(q, r))
Query: p
Query entails Knowledge Base: False
```

Date :
Page No :

Knowledge based Resolution

initialization:

1) Create an instance of knowledge Base class with an empty list of clauses:-

self.clauses = []

2) Adding a clause:-

append a new clause to the list of clauses in the knowledge Base.

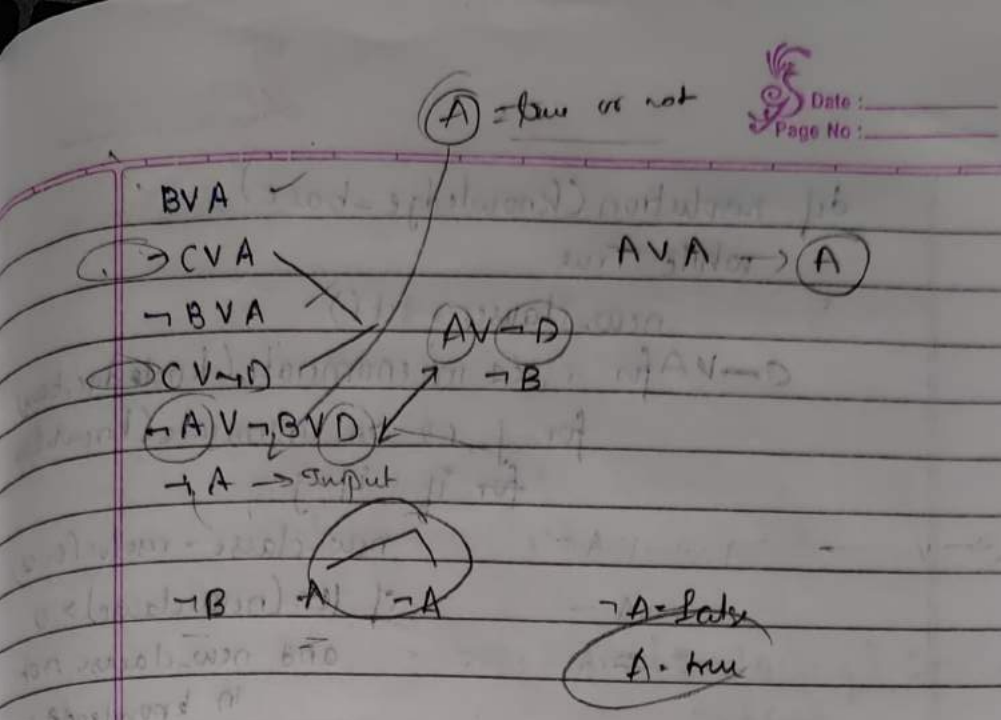
3) Resolving clause:-

Combine 2 clauses by common literals (eliminating complementary literals)

Negate the query & adds it to the knowledge base.

→ Repeatedly resolved the pairs of clauses in the knowledge base until a contradiction found or no new resolution are possible.

```
def resolve(self, clause_a, clause_b)
    return [literal for literal in clause_a
                    +clause_b)
    if (not' + literal) not in clause_a or
                    ↓
                literal not in clause_b)
```



Code :-

```
def negate_literal(literal)
    if literal[0] == '~':
        return literal[1:]
    else:
        return '~' + literal

def resolve(c1, c2)
    resolved_clause = set(c1) | set(c2)

    for literal in c2:
        if negate_literal in c2:
            resolved_clause.remove(literal)
            resolved_clause.remove(negate_literal
                                   (literal))

    return tuple(resolve_clause)
```

```
def resolution(knowledge_base)
    while True
        new_clauses = set()
        for i, c1 in enumerate(knowledge_base)
            for j, c2 in enumerate(knowledge_base)
                if i != j:
                    new_clause = resolve(c1, c2)
                    if len(new_clause) > 0
                        and new_clause not
                            in knowledge base
                        new_clause.add(new_
        if not new_clauses:
            break
        knowledge_base |= new_clauses
    return knowledge_base

if __name__ == "__main__":
    kb = {('p', 'q'), ('~p', 'r'), ('~q', 'r')}
    result = resolution(kb)
    print("original kB", kb)
    print("Resolved kB", result)
```

Date :
Page No :

→ Code for Entailment

```
def create_knowledge_base()
    p = symbols('p')
    q = symbols('q')
    r = symbols('r')

    knowledge_base = And(Implies(p,q), Implies(q,r),
                         Not(r))
    return knowledge_base

def query_entails(knowledge_base, query)
    entailment = satisfiable(And(knowledge_base,
                                 not(query)))
    return not entailment

if __name__ == "__main__":
    kb = create_knowledge_base()
    query = symbols('p')
    result = query_entails(kb, query)

    print("knowledge Base:", kb)
    print("Query", query)
    print("Query entails knowledge base:",
          result)
```

Output:

knowledge base : ~r & (Implies(p,q)) ∧ (Implies(q,r))

Query : p

Query entails knowledge base : False

OUTPUT:

```
Harika N (1BM21CS071)
['~P', 'R']
```

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

```
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.
```

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

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

9/1/24

Unification

Eg:- Knows (John, x) Knows (John, Jane)
{x/ Jane}

Step 1: If term 1 or term 2 is variable or constant, then:

a) term 1 or term 2 are identical
return NIL

b) Else if term1 is ~~avai~~ a variable
if term1 occurs in term 2
return FAIL
else
return {(term 2 / term1)}

c) else if term 2 is a variable
if term 2 occurs in term1
return FAIL
else
return (term1 / term2)

d) else return FAIL

Step 2:- If predicate(term1) ≠ predicate (term2)
return FAIL

Step 3: number of arguments ≠
return FAIL

Step 4: Set (SUBSET) to NIL

Step 5: For i=1 to the number of elements in term1
a) Call unify (ith term1, ith term2)
put result into S

b) S = FAIL
return FAIL

c) If S ≠ NIL
a. Apply S to the remainder of both L1 and L2
b. SUBST = APPEND (S, SUBST)

Step 6: Return SUBST

Proved

OUTPUT

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

```
Harika N (1BM21CS071)
Substitutions:
[('X', 'Richard')]
```

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

```
Harika N (1BM21CS071)

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

FOL to CNF conversion

Step 1: Create a list of SKOLEM-CONSTANTS

Step 2: Find ∀, ∃
If the attributes are lower case, replace them with a skolem constant.
remove used skolem constant or function from the list

If the attribute are both lowercase and uppercase replace the uppercase attribute with a skolem function.

Step 3: replace ⇔ with '-'
transform - as $Q \equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P)$

Step 4: replace ⇒ with '-'
$P \Rightarrow Q$ transform - as $\sim P \vee Q$

Step 5: Apply demorgan's law
replace $\sim[$
as $\sim P \wedge \sim Q$ (if | was present)

replace $\sim[$
as $\sim P | \sim Q$ if (& was present)

replace $\sim\sim$ with ''

OUTPUT

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

Harika N (1BM21CS071)

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

11/2024

Date :
Page No :

Forward chaining

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

Example:

KB

king(x) & greedy(x) ⇒ evil(x)
king(John)
greedy(John)
king(Richard)

Query

evil(x)

Code:

```
import re

def 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])})"

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.getVari-
                                     able()):
                if v:
                    constants[v] = fact.getConstants
                                   ()[i]
```

```
            new_lhs.append(fact)
        else:
            new_lhs.append(fact)
    return Implication(new_lhs, rhs)
else:
    return fact

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

    def tell(self, e):
        if '=>' in e:
            self.implications.add(Implication.from_string(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'{i+1}: {f}')

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

Output:

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

OUTPUT

```
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()
```

Harika N (1BM21CS071)

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