→ Ankit Kesar (1BM18CS150)

Pgm - 3

→ variable = {'P': 0, 'Q': 1, 'R': 2}

priority = {'~': 3, '^': 1, 'v': 2}

```
def - eval (i, val1, val2):
    if i == '^':
        return val2 and val1
    return val2 or val1

def isOperand (c):
    return c.isalpha() and c != 'v'

def isLeftParanthesis (c):
    return c == '('

def isRightParanthesis (c):
    return c == ')'.

def isStackEmpty (st):
    return len(st) == 0

def peekstack (stack)
    return stack[-1]

def hasLessOrEqualpriority (c1, c2):
    try: return priority[c1] <= priority[c2]
    except KeyError
        return false.
```

```
def toPostfix(infix)

    stack = []
    postfix = ''
    for c in infix:
        if isOperand(c)
            postfix += c

        elseif
            isleftParanthesis(c)
                stack.append c)
        elseif is RightParanthesis(c)
            operator = stack.pop()
            while not isleftParanthesis(operator):
                postfix += operator
                operata = stack.pop()

        else:
            while (not isEmpty(stack)) and hasSamerOrEqualPriority(c,peek
            stack)):
                postfix += stack.pop();
            returnpostfix

def evaluate(exp, comb):
    stack = []
    for i in exp:
        if isOperand(i):
            Stack.append(comb[variable(i)])
```

```
elif i == '~':
        val1 = stack.pop()
        stack.append(not val1)

return stack.pop()

def checkEntailment():
    kb = input('Enter the Knowledge Base: ')
    query = (input("Enter the query"));
combinations = [[True, T., T) (T, T, F)]

postfix_dp = toPostfix(kb)
postfix_q = toPostfix(query)

for comb in combinations:
    eval_kb = evaluatePostfix(postfix_b, combination)
    eval_q = evaluatepostfix(postfix_q, combination)
    print(combination, " : kb = ", eval_kb, " : q = ", eval_q);
    if (eval_kb == True)
        if (eval_q = False)
            print("Doesnot entails";
            return false
    print("The query entails the knowledgbase")
```