29/12/20

Chandan A.M.
1BM18CS025

Artificial Intelligence
LAB Test - 2

2. Consider P, Q and R as variables and KB as :
(P ∧ Q) ⇒ R ; (Q ⇒ P) ; Q

```
combinations = ((True, True, True), (True, True, False),
                (True, False, True), (True, False, False),
                (False, True, True), (False, True, False),
                (False, False, True), (False, False, False))
variable = {p = '0', q = '1', r = '2'}
priority = {'v' = 1, '^': 2, '~' = 3}
def input_~~all~~rules():
global kb, q
kb = input ("Enter rule: ")
 q = input ("Enter the query:'')

def entailment():
    global kb, q
    print ('*' * 10 + 'Truth Table Reference' + '*' * 10)
    print ('kb', 'alpha')
    print ('*' * 10)
```

1

[illegible]

Chandan A.M.
1BM18CS025

AI Lab Test - 2

```
for comb in combination :
    s = evaluatePostfix ( toPostfix (kb), comb)
    f = evaluate Postfix (toPostfix ( q), comb)
    print (s, t)
    print ('_' * 10)
    if s and not f :
        return ~~false~~ False
    return True

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

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

def isEmpty (stack) :
    return [illegible] len(stack) == 0
def isRightParanthesis (c) :
    return c == ')'
def peek (stack):
    return stack[-1]

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

[illegible]

Chandan A.M.
1BM18CS025

AI Lab Test - 2

```
def toPostfix (infix) :
        stack = []
        postfix = ""
        for c in infix :
            if isOperand (c) :
                postfix + = c
            ~~else~~ if isLeftParanthesis (c) :
                stack . append (c)
            elif isRightParanthesis (c) :
                operator = stack . pop ()
                while not isLeftParanthesis (operator) :
                    postive += operator
                    operator = stack . pop ( )

            else :
                while (not is Empty (stack) and
                       has Less Than Equal Priority (c, peak (Stack)))
                    postfix + = stack . pop ( )

            return postfix


def evaluatePostfix (exp , comb) :
        stack = []
        for i in exp :
            if isOperand (i) :
                stack . append ( comb [ variable [i] ])
            elif i == 'v' :
                val1 = stack . pop ()
                stack . append (not val1)
```

[illegible]

Chandan A. M.
1BM18CS025

AI Lab-2

```
else:
        val1 = stack.pop()
        val2 = stack.pop
        stack.append(_eval(i, val1, val2))
    return stack.pop()

def _eval(i, val1, val2):
    if i == '^'
        return val2 and val1
    return val2 or val1


~~input()~~
input_rules()
ans = entailment()
if ans:
    print("The KB entails query")
else:
    print("The KB does not entail query")
```

[illegible]