

## LAB TEST 2

Consider P,Q and R as variables and the Knowledge base contains following sentences:

$P \wedge Q \Rightarrow R$ ;  $(Q \Rightarrow P)$ ; Q

Design the code for TT entailment and show whether Knowledgebase entails R

### CODE:

```
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)]
#expand this set for more variables
variable={'p':0,'q':1, 'r':2}#expand this set matching combinations
indices for variables
#set of rules
kb=''#should be a cnf
q=''#should be a cnf
priority={'~':3,'v':1,'^':2}
```

```
def input_rules():
    global kb, q
    kb = (input("Enter Rule :"))
    q = input("Enter Query : ")

def entailment():
    global kb, q
    print('*' * 10 + "Truth Table Reference" + "*" * 10)
    print("kb", "alpha")
    print("*" * 10)
    for comb in combinations:
        s = evaluatePostfix(toPostfix(kb), comb)
        f = evaluatePostfix(toPostfix(q), comb)
        print(s, f)
        print("-" * 10)
        if s and not f:
            return False
    return True

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

```

def isLeftParenthesis(c):
    return c == "("

def isRightParenthesis(c):
    return c == ")"

def isEmpty(stack):
    return len(stack) == 0

def peek(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
        else:
            if isLeftParenthesis(c):
                stack.append(c)
            elif isRightParenthesis(c):
                operator = stack.pop()
                while not isLeftParenthesis(operator):
                    postfix += operator
                    operator = stack.pop()
            else:
                while (not isEmpty(stack)) and hasLessOrEqualPriority(c,
peek(stack)):
                    postfix += stack.pop()
                stack.append(c)
    while (not isEmpty(stack)):
        postfix += stack.pop()
    return postfix

```

```

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

def evaluatePostfix(exp, comb):
    stack = []
    for i in exp:
        if isOperand(i):
            stack.append(comb[variable[i]])
        elif i == '~':
            val1 = stack.pop()
            stack.append(not val1)
        else:
            val1 = stack.pop()
            val2 = stack.pop()
            stack.append(_eval(i, val2, val1))
    return stack.pop()

```

## OUTPUT:

```

In [7]: # main code
input_rules()
ans = entailment()
if ans: print("The Knowledge Base Entails Query")
else: print("The Knowledge Base Doesn't Entail Query")

```

```

Enter Rule : (~pv~qvr)^(~qvp)^q
Enter Query : r
*****Truth Table Reference*****
kb alpha
*****
True True
-----
False False
-----
False True
-----
False False
-----
False True
-----
False False
-----
False True
-----
False False
-----
False True
-----
False False
-----
The Knowledge Base Entails Query

```

converting the given facts to CNF form

$(P \wedge Q) \Rightarrow R \wedge (Q \Rightarrow P) \wedge Q$  becomes  $(pvqvr)^(~qvp)^q$

Github: <https://github.com/kp10x/AI-Lab/tree/master/AI%20LAB%20TEST%202>