29.12.12

AI Lab

Krishna prasad V
1BM18CS047
5A

Q. Check if query entails knowledge base using Knowledge base truth tables.

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)]
                                                          // 8 combis since
                                                                3 vals

variables = {'p':0, 'q':1, 'r'}          // all variables in the kb
  kb = ''
  q = ''
  priority = {'~':3, 'v':1, '^':2}

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

def entailment():
    global kb, q
    [redacted]
    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)
        if s and not f:
            return False
    return True

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

def is Left Paranthesis (c) :
    return c == "("

def is Right Paranthesis (c):
    return c == ")"

def is Empty (stack):
    return len (stack) ==0

def peek (stack):
    return stack [-1]

def has Less Or Equal Priority (c1,c2):
    try: return priority [c1] <= priority [c2]
    except KeyError : return False

def to Postfix (infix) :
    stack = []
    postfix = ''
    for c in infix :
        if is Operand (c):
            postfix += c

        else :
            if isLeft Paranthesis (c):
                stack.append (c)
        elif isRightParanthesis(c) :
            operator = stack.pop()
            while not isLeftParanthesis(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()
```