AI - LAB - TEST-2

VYSHNAVI.M.V
1BM18CS125
5'C' C2

combinations :

[(True, True), (True, False),
(False, False), (False, True)]

variable = {'s': 0, 't': 1}

```
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
    print('Truth Table')
    print('kb', 'alpha')
    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 toPostfix(infix):
    stack = []
    postfix = ""
    for c in ~~postfix~~ infix:
        if isOperand(c): postfix += c
        else:
            if isLeftParanthesis(c):
                stack.append(c)
            elif isRightParanthesis(c):
                operator = stack.pop()
                while not isLeftParanthesis(op):
                    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 evaluatePostfix(exp, comb):
    stack = []
    for i in exp:
        if isOperand(i):
            stack.append(comb[variable[i]])
        elif i == '~':
            val1 = stack.pop()
            val2 = stack.pop()
            stack.append(_eval(i, val2, val1))
    return stack.pop()

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

input_rules()
ans = entailment()

if ans:
    print("Kb entails query")

else:
    print("Kb doesn't entail query")
```