29/12/2020

AI LAB TEST - 2

K. ARUN TEJA
1BM18CS041
5A [A2]

2) Consider P, Q, R as variables and knowledge base contains following sentences:

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

Design code for TT entailment and show whether knowledge base entails R.

A.)

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

 ~~def inputevalues():~~
 def input_rules():                          # (~P v ~Q v R)^(~Q v P)^Q
     global kb, q                            [illegible]
     kb = input("Enter rule: ")
      q = input("Enter query: ")

 def entailment():
     global kb, q
     print("Truth table Reference")
     print("P", "Q", "R", "KB", 'Alpha')
      print("_" * 20)
     for comb in combinations:
             S = evaluatePostfix(toPostfix(kb), comb)
             f = evaluatePostfix(toPostfix(q), comb)

             a, b, c, = comb
             print(a, b, c, s, f)
             print("_" * 10)
             if s and not f:
                     return false

     return True
```

[illegible]

K.ARUN TEJA
1BM18CS041
5A [A2]

```
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)):
                        post fix += stack.pop()
                    Stack.append(c)
        while (not isEmpty(stack)):
            postfix += stack.pop()
        return postfix
```

[illegible]

K. ARUN TEJA
BM18CS041
5A [A2]

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

Input_rules()
ans = entailment ()
if ans: print (" The knowledge base entails query ")
else: print (" The knowledge base doesn't entails Query ")
```

[illegible]