Lab 6:checks if a query is entrained by a knowledge base(KB) using propositional logic and truth table

import math

# Define all possible truth assignments for the variables

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)

]

# Map propositional variables to index in the combination tuple

variables = {'p': 0, 'q': 1, 'r': 2}

# Define operator precedence

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

# Global variables for the knowledge base and the query

kb = ''

q = ''

# Input rules for KB and Query

def input\_rules():

global kb, q

kb = input("Enter rule (Knowledge Base): ")

q = input("Enter the Query: ")

# Check if the knowledge base entails the query

def entailment():

global kb, q

print('\*' \* 10 + "Truth Table Reference" + '\*' \* 10)

print('KB', 'Query')

print('\*' \* 10)

for comb in combinations:

# Evaluate both the KB and query using the current truth assignment

kb\_result = evaluatePostfix(toPostfix(kb), comb)

query\_result = evaluatePostfix(toPostfix(q), comb)

# Print the results of the KB and query for this combination

print(kb\_result, query\_result)

print('-' \* 10)

# If KB is true and query is false, it doesn't entail the query

if kb\_result and not query\_result:

return False

return True # If no counterexample, KB entails the query

# Check if a character is an operand (variable)

def isOperand(c):

return c.isalpha() and c != 'v'

# Check for left parenthesis

def isLeftParanthesis(c):

return c == '('

# Check for right parenthesis

def isRightParanthesis(c):

return c == ')'

# Check if the stack is empty

def isEmpty(stack):

return len(stack) == 0

# Peek at the top element of the stack

def peek(stack):

return stack[-1]

# Check if the first operator has less or equal priority than the second operator

def hasLessOrEqualPriority(c1, c2):

try:

return priority[c1] <= priority[c2]

except KeyError:

return False

# Convert infix expression to postfix

def toPostfix(infix):

stack = []

postfix = ''

for c in infix:

if isOperand(c):

postfix += c

elif isLeftParanthesis(c):

stack.append(c)

elif isRightParanthesis(c):

operator = stack.pop()

while not isLeftParanthesis(operator):

postfix += operator

operator = stack.pop()

else: # operator ('~', 'v', '^')

while not isEmpty(stack) and hasLessOrEqualPriority(c, peek(stack)):

postfix += stack.pop()

stack.append(c)

while not isEmpty(stack):

postfix += stack.pop()

return postfix

# Evaluate the postfix expression for a given combination of truth values

def evaluatePostfix(exp, comb):

stack = []

for i in exp:

if isOperand(i):

# Get the truth value for the variable from the combination

stack.append(comb[variables[i]])

elif i == '~':

# Negation: Pop the top value, negate it and push back

val1 = stack.pop()

stack.append(not val1)

else:

# And ('^') or Or ('v') operator

val1 = stack.pop()

val2 = stack.pop()

stack.append(\_eval(i, val2, val1))

return stack.pop()

# Evaluate logical AND or OR operation

def \_eval(operator, val1, val2):

if operator == '^': # AND operation

return val2 and val1

elif operator == 'v': # OR operation

return val2 or val1

# Main execution flow

def main():

input\_rules() # Get the KB and query from the user

result = entailment() # Check entailment

if result:

print("The Knowledge Base entails the query.")

else:

print("The Knowledge Base does not entail the query.")

# Run the main function

if \_\_name\_\_ == "\_\_main\_\_":

main()

output:



