Week-6  
Create a knowledge base using propositional logic and show that the given query entails the knowledge base or not.

from itertools import product

def pl\_true(sentence, model):

    """Evaluates if a sentence is true in a given model."""

    if isinstance(sentence, str):

        return model.get(sentence, False)

    elif isinstance(sentence, tuple) and len(sentence) == 2:  # NOT operation

        operator, operand = sentence

        if operator == "NOT":

            return not pl\_true(operand, model)

    elif isinstance(sentence, tuple) and len(sentence) == 3:

        operator, left, right = sentence

        if operator == "AND":

            return pl\_true(left, model) and pl\_true(right, model)

        elif operator == "OR":

            return pl\_true(left, model) or pl\_true(right, model)

        elif operator == "IMPLIES":

            return not pl\_true(left, model) or pl\_true(right, model)

        elif operator == "IFF":

            return pl\_true(left, model) == pl\_true(right, model)

def tt\_entails(kb, alpha, symbols):

    """Checks if KB entails alpha using truth-table enumeration."""

    all\_models = product([False, True], repeat=len(symbols))

    valid\_models = []

    for values in all\_models:

        model = dict(zip(symbols, values))

        kb\_value = pl\_true(kb, model)

        alpha\_value = pl\_true(alpha, model)

        if kb\_value:  # If KB is true in this model

            if not alpha\_value:  # If KB is true but α is not, entailment fails

                return False, None

            else:

                valid\_models.append(model)

    return True, valid\_models

def print\_truth\_table(kb, alpha, symbols):

    """Generates and prints the truth table for KB and α."""

    headers = ["A      ", "B      ", "C      ", "A ∨ C  ", "B ∨ ¬C ", "KB     ", "α     "]

    print(" | ".join(headers))

    print("-" \* (len(headers) \* 9))  # Separator line

    # Generate all combinations of truth values

    for values in product([False, True], repeat=len(symbols)):

        model = dict(zip(symbols, values))

        # Evaluate sub-expressions and main expressions

        a\_or\_c = pl\_true(("OR", "A", "C"), model)

        b\_or\_not\_c = pl\_true(("OR", "B", ("NOT", "C")), model)

        kb\_value = pl\_true(kb, model)

        alpha\_value = pl\_true(alpha, model)

        # Print the truth table row

        row = values + (a\_or\_c, b\_or\_not\_c, kb\_value, alpha\_value)

        row\_str = " | ".join(str(v).ljust(7) for v in row)

        # Highlight rows where both KB and α are true

        if kb\_value and alpha\_value:

            print(f"\033[92m{row\_str}\033[0m")  # Green color for rows where KB and α are true

        else:

            print(row\_str)

# Define the knowledge base and query

symbols = ["A", "B", "C"]

kb = ("AND", ("OR", "A", "C"), ("OR", "B", ("NOT", "C")))

alpha = ("OR", "A", "B")

# Print the truth table

print\_truth\_table(kb, alpha, symbols)

# Run the truth-table entailment check

entailment, models = tt\_entails(kb, alpha, symbols)

# Print the result

print("\nResult:")

if entailment:

    print("KB entails α.")

    print("The values of A, B, C for which KB and α are true:")

    for model in models:

        print(model)

else:

    print("KB does not entail α.")

OUTPUT:

