```python
import pandas as pd
from itertools import product
import re

def tokenize(sentence):
    # Tokenize based on logical operators, parentheses, and symbols
    # Tokens are: '(', ')', 'and', 'or', 'not', variables (letters),
whitespace ignored
    token_pattern = r'\w+|[()]'
    return re.findall(token_pattern, sentence)

def pl_true(sentence, model):
    # Tokenize the sentence
    tokens = tokenize(sentence)

    # Logical operators in Python are lowercase
    logical_ops = {'and', 'or', 'not'}

    evaluated_tokens = []
    for token in tokens:
        token_lower = token.lower()
        if token_lower in logical_ops:
            # Keep operators as is (lowercase)
            evaluated_tokens.append(token_lower)
        elif token in model:
            # Replace symbol with 'True' or 'False' string
            evaluated_tokens.append(str(model[token]))
        else:
            # Parentheses or unknown tokens are kept as is
            evaluated_tokens.append(token)

    # Join tokens with spaces for safe eval
    eval_sentence = ' '.join(evaluated_tokens)

    try:
        return eval(eval_sentence)
    except Exception as e:
        print(f"Error evaluating sentence: {eval_sentence}")
        raise e

def tt_entails(kb, alpha, symbols):
    truth_table = []

    for model in product([False, True], repeat=len(symbols)):
        model_dict = dict(zip(symbols, model))

        kb_value = pl_true(kb, model_dict)
        alpha_value = pl_true(alpha, model_dict)
```

```
        row = {
            'A': model_dict.get('A', False),
            'B': model_dict.get('B', False),
            'C': model_dict.get('C', False),
            'A or C': model_dict.get('A', False) or model_dict.get('C',
False),
            'B or not C': model_dict.get('B', False) or not
model_dict.get('C', False),
            'KB': kb_value,
            'α': alpha_value
        }
        truth_table.append(row)

        if kb_value and not alpha_value:
            return False, pd.DataFrame(truth_table)

    return True, pd.DataFrame(truth_table)

def get_symbols(kb, alpha):
    # Find unique uppercase letters as symbols
    return sorted(set(re.findall(r'[A-Z]', kb + alpha)))

# Example usage:

kb = "(A or C) and (B or not C)"
alpha = "A or B"

symbols = get_symbols(kb, alpha)

result, truth_table = tt_entails(kb, alpha, symbols)

print("Truth Table:")
display(truth_table)

if result:
    print("\nKB entails α")
else:
    print("\nKB does not entail α")
```

Page No.:
Date:
YOUVA

⇒ Propositional logic.

P	Q	¬P	P∧Q	P∨Q	P ⟺ Q
false	false	true	false	false	true
false	true	true	true	true	~~true~~ false
true	false	false	true	true	false
true	true	false	true	true	true

Implementation of TT enumeration algorithm for deciding propositional entailment.

eg: $\alpha = A \vee B$

$KB = (A \vee B) \wedge (B \vee \neg C)$

Checking that $KB \models \alpha$

A	B	C	A∨B	B∨¬C	KB	α
false	false	false	false	true	false	false
false	false	true	true	false	false	false
false	true	false	false	true	false	true
false	true	true	true	true	true	true
true	false	false	true	true	true	true
true	false	true	true	false	false	true
true	true	false	true	true	true	true
true	true	true	true	true	true	true

function TT-Entails? (KB, α) returns true or false. inputs: KB, the knowledge base, a sentence in propositional logic α, the query, a sentence in propositional logic

symbols ← a list of the proposition symbols in KB & α.
return TT-CHECK-ALL(KB, α, symbols, { })

function TT-CHECK-ALL (KB, α, symbols, model) return T or F.

if EMPTY? (symbols) then.
 if PL-TRUE? (KB, model) then
  return PL-TRUE? (α, model).
 else return true.
else do.
 P ← FIRST (symbols)
 rest ← REST (symbols)
return (TT-CHECK-ALL (KB, α, rest, model ∪ { P = true }))
 and
 TT-CHECK-ALL (KB, α, rest, model ∪ { P = false }))

a entails b → false.

Ex:- KB = ~(S/T)
 α = S & T

S	T	KB	α
0	0	1	0
0	1	0	0
1	0	0	0
1	1	0	1

a entails c ; true.

Propositional Logic 1BM23CS316.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

[1] ✓ 0s

```
def get_symbols(kb, alpha):
    # Find unique uppercase letters as symbols
    return sorted(set(re.findall(r'[A-Z]', kb + alpha)))

# Example usage:

kb = "(A or C) and (B or not C)"
alpha = "A or B"

symbols = get_symbols(kb, alpha)

result, truth_table = tt_entails(kb, alpha, symbols)

print("Truth Table:")
display(truth_table)

if result:
    print("\nKB entails α")
else:
    print("\nKB does not entail α")
```

```
Truth Table:
```

	A	B	C	A or C	B or not C	KB	α
0	False	False	False	False	True	False	False
1	False	False	True	True	False	False	False
2	False	True	False	False	True	False	True
3	False	True	True	True	True	True	True
4	True	False	False	True	True	True	True
5	True	False	True	True	False	False	True
6	True	True	False	True	True	True	True
7	True	True	True	True	True	True	True

```
KB entails α
```