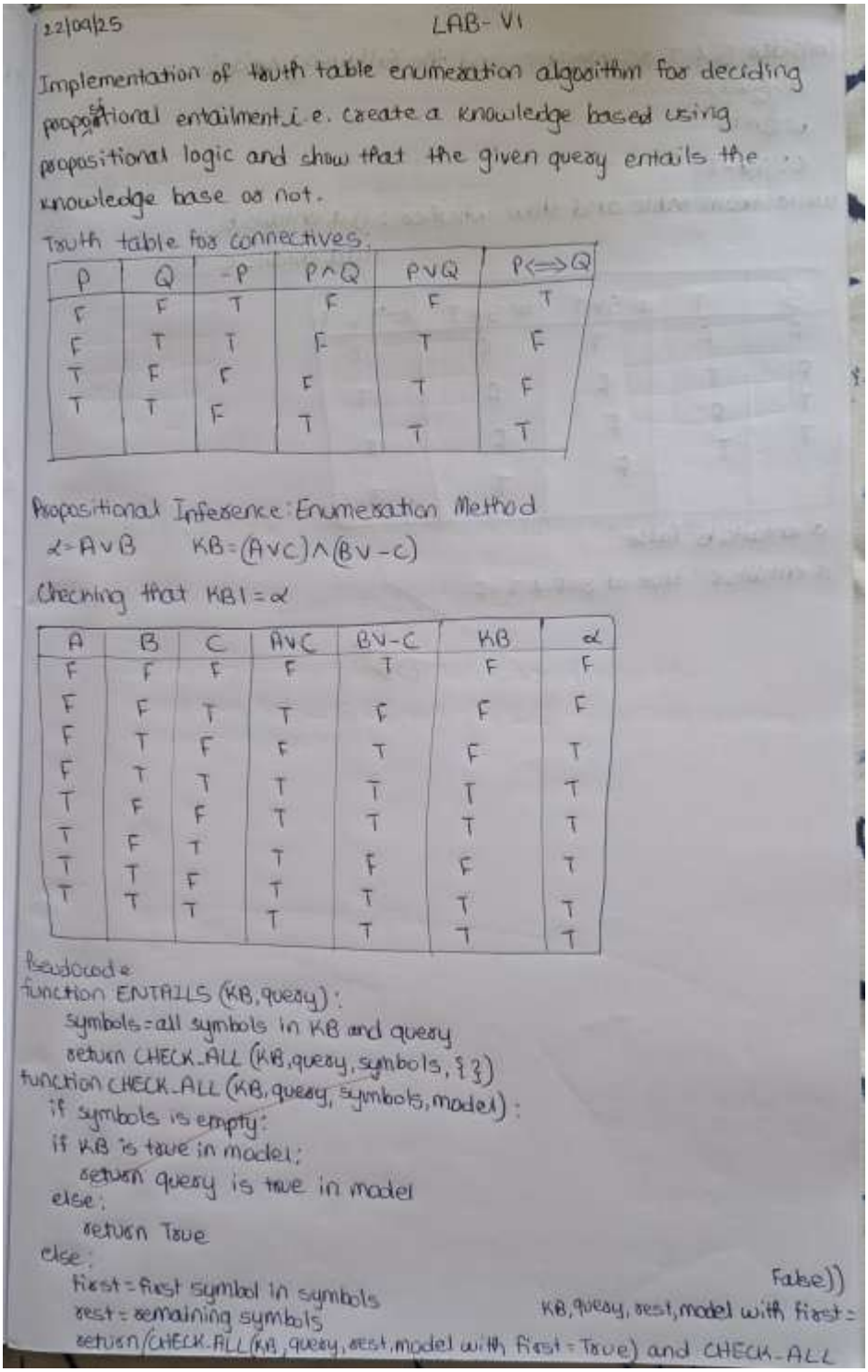
Lab 6

Implement truth table enumeration algorithm for deciding propositional entailment.

Algorithm:



22/09/25 LAB- VI

Implementation of truth table enumeration algorithm for deciding propositional entailment i.e. create a knowledge based using propositional logic and show that the given query entails the knowledge base or not.

Truth table for connectives:

P	Q	¬P	P∧Q	P∨Q	P⟺Q
F	F	T	F	F	T
F	T	T	F	T	F
T	F	F	F	T	F
T	T	F	T	T	T

Propositional Inference: Enumeration Method

α = A∨B KB = (A∨C)∧(B∨¬C)

Checking that KB ⊨ α

A	B	C	A∨C	B∨¬C	KB	α
F	F	F	F	T	F	F
F	F	T	T	F	F	F
F	T	F	F	T	F	T
F	T	T	T	T	T	T
T	F	F	T	T	T	T
T	F	T	T	F	F	T
T	T	F	T	T	T	T
T	T	T	T	T	T	T

Pseudocode

function ENTAILS(KB, query):
  symbols = all symbols in KB and query
  return CHECK-ALL(KB, query, symbols, {})

function CHECK-ALL(KB, query, symbols, model):
  if symbols is empty:
    if KB is true in model:
      return query is true in model
    else:
      return True
  else:
    first = first symbol in symbols
    rest = remaining symbols
    return CHECK-ALL(KB, query, rest, model with first = True) and CHECK-ALL(KB, query, rest, model with first = False))

Consider S & T as variables and the following relations:

a:-(S v T)

b: (S ∧ T)

c: T v -T

Write truth table and show whether : i) a entails b

ii) a entails c

S	T	a = S v T	b = S ∧ T	c = T v -T
F	F	T	F	T
F	T	F	F	T
T	F	F	F	T
T	T	F	T	T

a entails b : false

a entails c : true at S = F & T = F

22/09

Code:

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

def tokenize(sentence):
    # Now also tokenize symbols like ∨ (logical OR)
    # Added ∨, ∧, ¬ in the regex to separate them as tokens
    token_pattern = r'\w+|[()∨∧¬]'
    return re.findall(token_pattern, sentence)

def pl_true(sentence, model):
    tokens = tokenize(sentence)
    logical_ops = {'and', 'or', 'not', '∨', '∧', '¬'}
    evaluated_tokens = []
    for token in tokens:
        if token == '∨':
            evaluated_tokens.append('or')  # replace symbol with python 'or'
        elif token == '∧':
            evaluated_tokens.append('and')  # replace symbol with python 'and'
        elif token == '¬':
            evaluated_tokens.append('not')  # replace symbol with python 'not'
        elif token.lower() in logical_ops:
            evaluated_tokens.append(token.lower())
        elif token in model:
            evaluated_tokens.append(str(model[token]))
        else:
```

```
            evaluated_tokens.append(token)
        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 ∨ C': model_dict.get('A', False) or model_dict.get('C', False),
            'B ∨ ¬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):
    return sorted(set(re.findall(r'[A-Z]', kb + alpha)))

# Example usage:
kb = "(A ∨ C) ∧ (B ∨ ¬C)"
alpha = "A ∨ B"
symbols = get_symbols(kb, alpha)
result, truth_table = tt_entails(kb, alpha, symbols)

def highlight_kb_alpha(row):
    if row['KB'] and row['α']:
        return ['background-color: lightgreen' if col in ['KB', 'α'] else '' for col in row.index]
    else:
        return ['' for _ in row.index]

print("Shreya Raj 1BM23CS317")
styled_table = truth_table.style.apply(highlight_kb_alpha, axis=1)
display(styled_table)

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

Output:

Shreya Raj 1BM23CS317

	A	B	C	A ∨ C	B ∨ ¬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