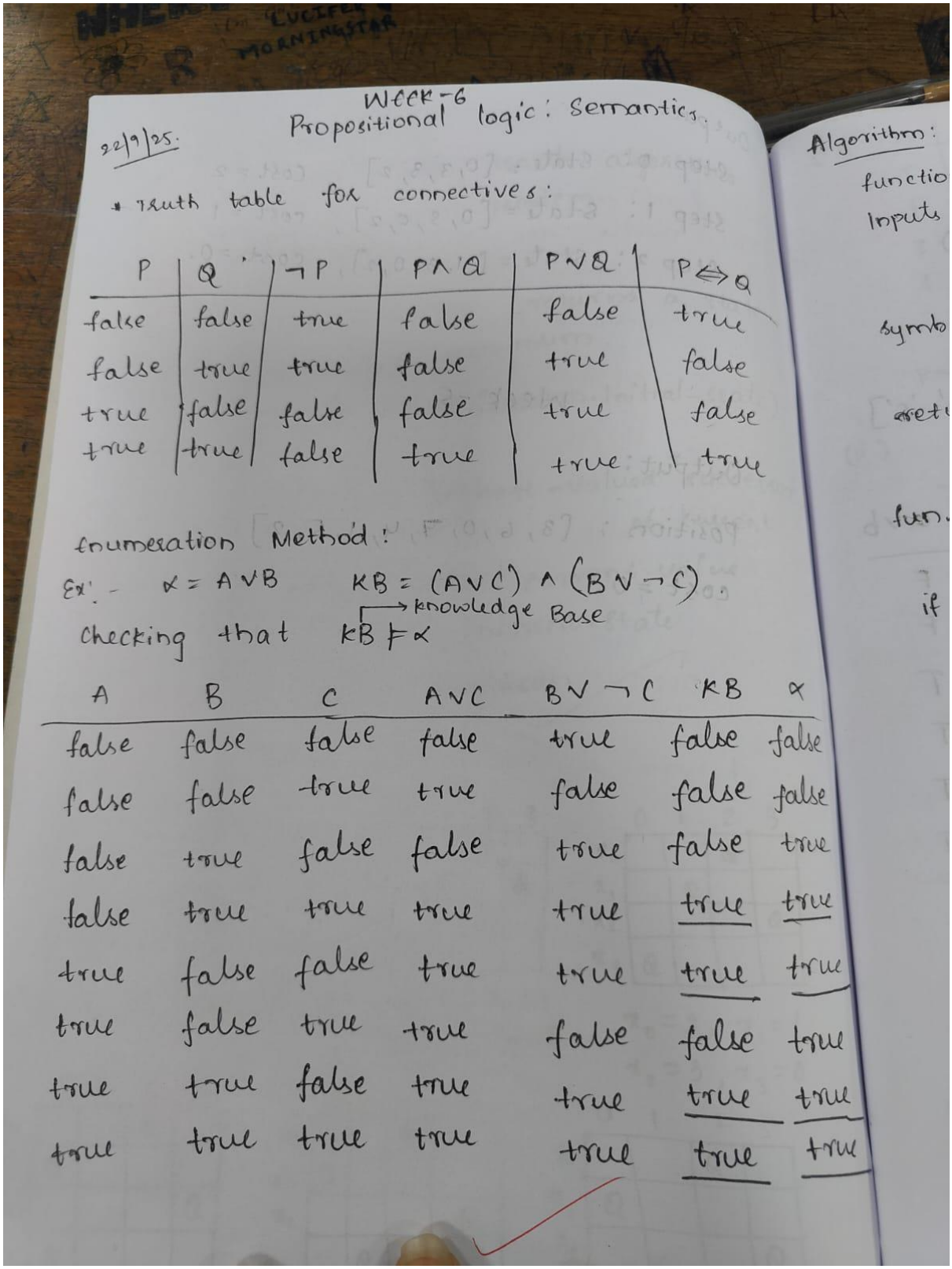
Week - 6

Propositional Logic: Semantic

Algorithm:



Week-6

22/9/25.

Propositional logic: Semantics.

* Truth table for connectives:

P	Q	¬P	P∧Q	P∨Q	P⇔Q
false	false	true	false	false	true
false	true	true	false	true	false
true	false	false	false	true	false
true	true	false	true	true	true

Enumeration Method:

Ex:- $\alpha = A \vee B$ $KB = (A \vee C) \wedge (B \vee \neg C)$

KB → knowledge Base

Checking that $KB \models \alpha$

A	B	C	A∨C	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

Algorithm:

functio

Inputs

symb

fun

if

Algorithm:

function TT-Entails? (KB, α) returns true or false
Inputs : KB, the knowledge base
α, the query

symbols ← a list of the proposition symbols in KB and α.
return TT-check-All (KB, α, symbols, {})

function TT-Check-All (KB, α, symbols, model)
return true or false
if EMPTY? (symbols) then
if PL-TRUE? (KB, 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}))

Output:

Enter proportional formulas in kb:
(a v c), (b v -c)
Enter combination of KB (AND/OR):
AND
Enter the query formula:
a v b
Proposition variables found: ['a', 'b', 'c']

Truth Table:

a	b	c	(a v c)	(b v -c)	KB	a v b
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

Knowledge Base entails the query: YES

Consider: S & T as variables and following relation:

a: ¬(s v t) b: (s ∧ t) c: T v ¬T.

Write TT and show whether

i) a entails b

ii) a v c

(i)

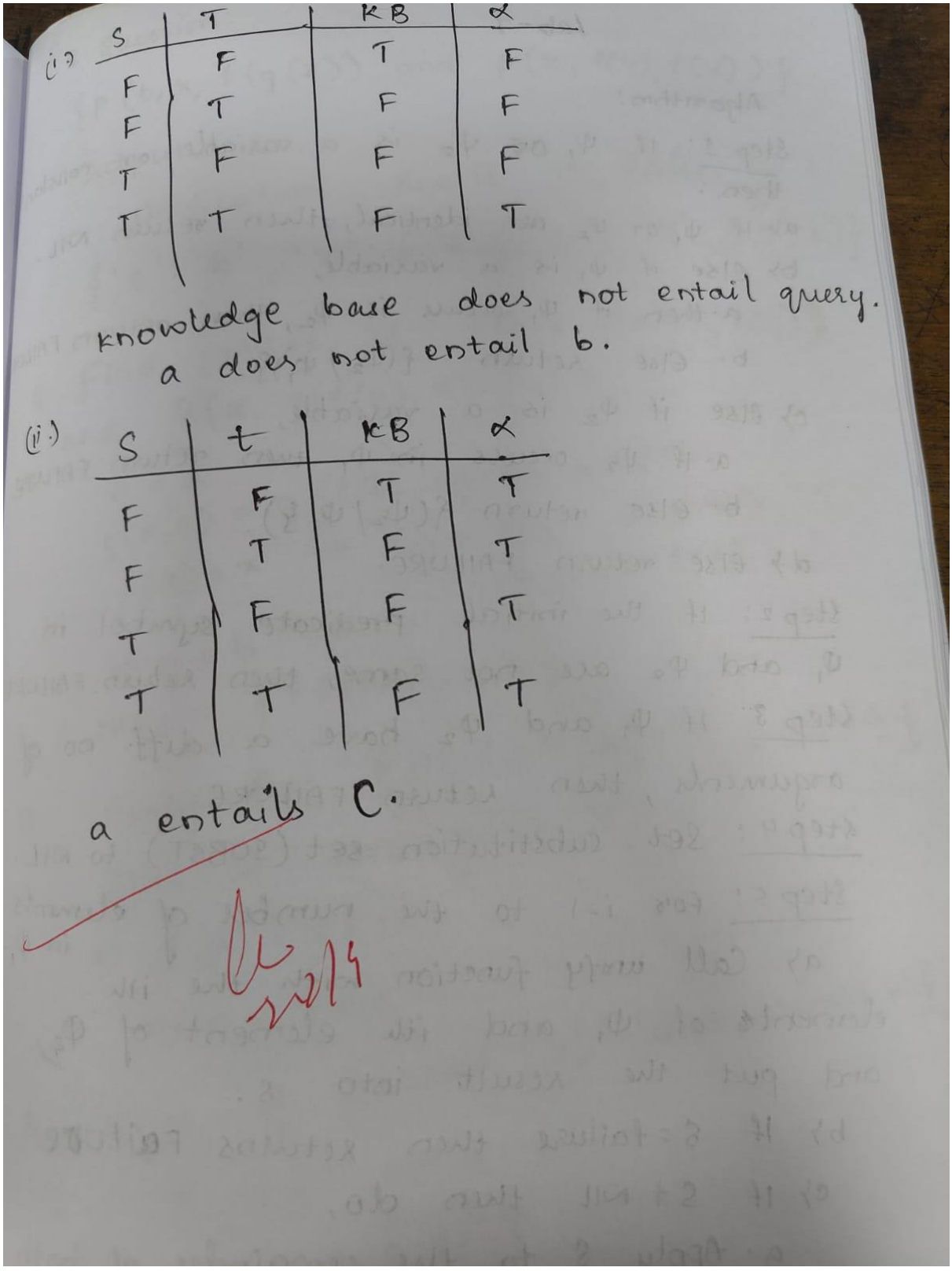
S
F
F
T
T

know

(ii)

S
F
F
T

a



(i)

S	T	KB	α
F	F	T	F
F	T	F	F
T	F	F	F
T	T	F	T

knowledge base does not entail query.
a does not entail b.

(ii)

S	t	KB	α
F	F	T	T
F	T	F	T
T	F	F	T
T	T	F	T

a entails C.

Output:

```
Enter propositional formulas in the knowledge base (separated by commas):
(a v c), (b v -c)
Enter how to combine KB formulas (AND / OR):
AND
Enter the query formula:
a v b

Propositional Variables found: ['A', 'B', 'C']

Full Truth Table:

   A    |    B    |    C    | (a v c) | (b v -c) | KB (AND) |  a v b
-------------------------------------------------------------------
   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

Knowledge Base entails the query: YES
Sareddy Poojya Sree
1BM23CS303
```

Code:

```python
import itertools

def preprocess_formula(formula: str) -> str:
    formula = formula.upper()
    formula = formula.replace('^', '&')
    formula = formula.replace('V', '|')
    formula = formula.replace('v', '|')
    formula = formula.replace('-', '~')
    return formula

def tokenize(s):
    tokens = []
    i = 0
    while i < len(s):
        c = s[i]
        if c == ' ':
            i += 1
            continue
        if c in ('(', ')', '~', '&', '|'):
            tokens.append(c)
            i += 1
        elif c == '-':
            if i+1 < len(s) and s[i+1] == '>':
                tokens.append('->')
                i += 2
            else:
```

```
                raise ValueError("Invalid token: '-' not followed by
'>'")
        elif c == '<':
            if s[i:i+3] == '<->':
                tokens.append('<->')
                i += 3
            else:
                raise ValueError("Invalid token starting with '<'")
        elif c.isalpha() and c.isupper():
            tokens.append(c)
            i += 1
        else:
            raise ValueError(f"Invalid character: {c}")
    return tokens

def parse_formula(tokens):
    def parse_expr():
        return parse_biconditional()
    def parse_biconditional():
        left = parse_implication()
        while tokens and tokens[0] == '<->':
            tokens.pop(0)
            right = parse_implication()
            left = ('<->', left, right)
        return left
    def parse_implication():
        left = parse_or()
        while tokens and tokens[0] == '->':
            tokens.pop(0)
            right = parse_implication()
            left = ('->', left, right)
        return left
    def parse_or():
        left = parse_and()
        while tokens and tokens[0] == '|':
            tokens.pop(0)
            right = parse_and()
            left = ('|', left, right)
        return left
    def parse_and():
        left = parse_not()
        while tokens and tokens[0] == '&':
            tokens.pop(0)
            right = parse_not()
            left = ('&', left, right)
        return left
    def parse_not():
        if tokens and tokens[0] == '~':
```

```
            tokens.pop(0)
            operand = parse_not()
            return ('~', operand)
        else:
            return parse_atom()
    def parse_atom():
        if not tokens:
            raise ValueError("Unexpected end of input")
        token = tokens.pop(0)
        if token == '(':
            node = parse_expr()
            if not tokens or tokens.pop(0) != ')':
                raise ValueError("Missing closing parenthesis")
            return node
        elif token.isalpha():
            return ('var', token)
        else:
            raise ValueError(f"Unexpected token: {token}")
    return parse_expr()

def eval_parsed_formula(node, valuation):
    op = node[0]
    if op == 'var':
        return valuation[node[1]]
    elif op == '~':
        return not eval_parsed_formula(node[1], valuation)
    elif op == '&':
        return eval_parsed_formula(node[1], valuation) and
eval_parsed_formula(node[2], valuation)
    elif op == '|':
        return eval_parsed_formula(node[1], valuation) or
eval_parsed_formula(node[2], valuation)
    elif op == '->':
        return (not eval_parsed_formula(node[1], valuation)) or
eval_parsed_formula(node[2], valuation)
    elif op == '<->':
        return eval_parsed_formula(node[1], valuation) ==
eval_parsed_formula(node[2], valuation)
    else:
        raise ValueError(f"Unknown node type: {op}")

def extract_vars(formulas):
    vars = set()
    def helper(node):
        if node[0] == 'var':
            vars.add(node[1])
        elif node[0] in ('~', '&', '|', '->', '<->'):
            if isinstance(node[1], tuple):
```

```
                helper(node[1])
            if len(node) > 2 and isinstance(node[2], tuple):
                helper(node[2])
    for f in formulas:
        helper(f)
    return sorted(vars)

def main():
    print("Enter propositional formulas in the knowledge base
(separated by commas):")
    kb_input = input().strip()
    kb_raw = [f.strip() for f in kb_input.split(',') if f.strip()]
    kb_processed = [preprocess_formula(f) for f in kb_raw]

    print("Enter how to combine KB formulas (AND / OR):")
    comb = input().strip().upper()
    if comb not in ('AND', 'OR'):
        print("Invalid combination choice, defaulting to AND.")
        comb = 'AND'

    print("Enter the query formula:")
    query_input = input().strip()
    query_processed = preprocess_formula(query_input)

    kb_parsed = []
    for formula in kb_processed:
        tokens = tokenize(formula)
        parsed = parse_formula(tokens)
        if tokens:
            raise ValueError("Extra tokens after parsing formula")
        kb_parsed.append(parsed)

    tokens_query = tokenize(query_processed)
    query_parsed = parse_formula(tokens_query)
    if tokens_query:
        raise ValueError("Extra tokens after parsing query")

    all_formulas = kb_parsed + [query_parsed]
    vars = extract_vars(all_formulas)

    print("\nPropositional Variables found:", vars)

    col_width = 7
    headers = vars + kb_raw + [f'KB ({comb})', query_input]
    print("\nFull Truth Table:\n")
    print(" | ".join(h.center(col_width) for h in headers))
    print("-" * (len(headers) * (col_width + 3) - 3))
```

```
    entails = True
    for values in itertools.product([False, True], repeat=len(vars)):
        valuation = dict(zip(vars, values))
        kb_vals = [eval_parsed_formula(f, valuation) for f in
kb_parsed]

        if comb == 'AND':
            kb_combined = all(kb_vals)
        else:
            kb_combined = any(kb_vals)

        query_val = eval_parsed_formula(query_parsed, valuation)

        row_vals = [('T' if val else 'F').center(col_width) for val in
values]
        row_vals += [('T' if val else 'F').center(col_width) for val in
kb_vals]
        row_vals.append(('T' if kb_combined else
'F').center(col_width))
        row_vals.append(('T' if query_val else 'F').center(col_width))

        print(" | ".join(row_vals))

        # Check entailment condition:
        if kb_combined and not query_val:
            entails = False

    print("\nKnowledge Base entails the query:" , "YES" if entails else
"NO")

if __name__ == "__main__":
    main()

print("Sareddy Poojya Sree\n1BM23CS303")
```