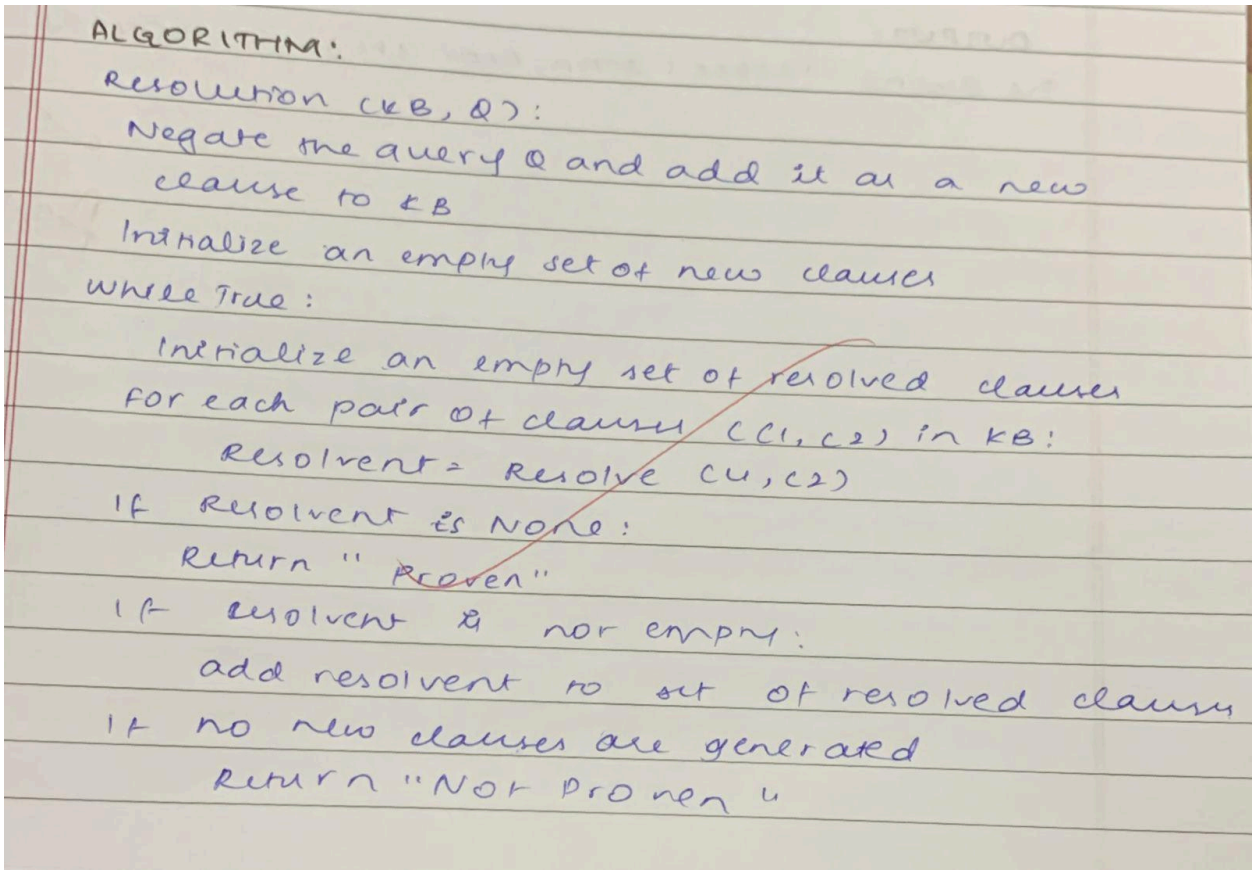
03-12-2024 WEEK-8

KNOWLEDGE BASE USING RESOLUTION

ALGORITHM-



ALGORITHM:

Resolution (KB, Q):

Negate the query Q and add it as a new clause to KB

Initialize an empty set of new clauses

While True:

Initialize an empty set of resolved clauses

For each pair of clauses (C1, C2) in KB:

Resolvent = Resolve (C1, C2)

If Resolvent is None:

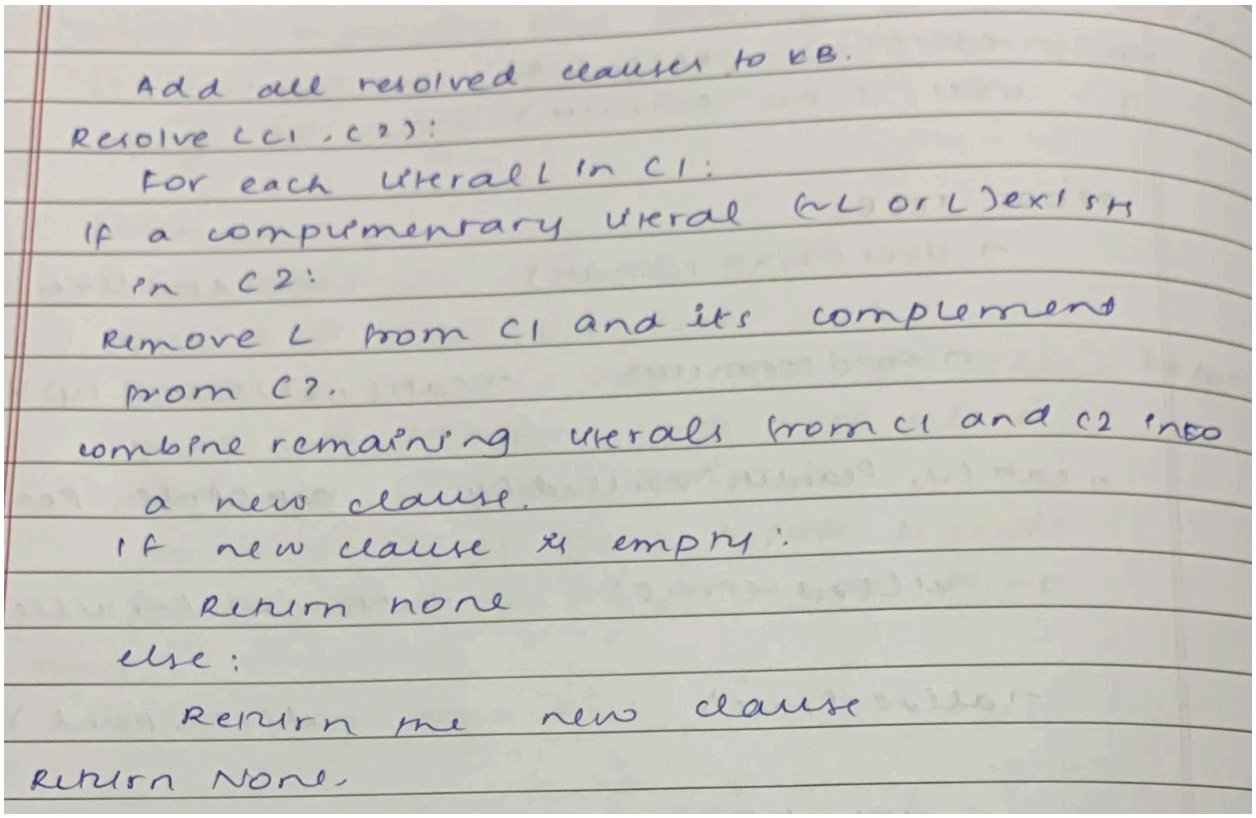
Return "Proven"

If Resolvent is not empty:

add resolvent to set of resolved clauses

If no new clauses are generated

Return "Not Proven"



CODE-

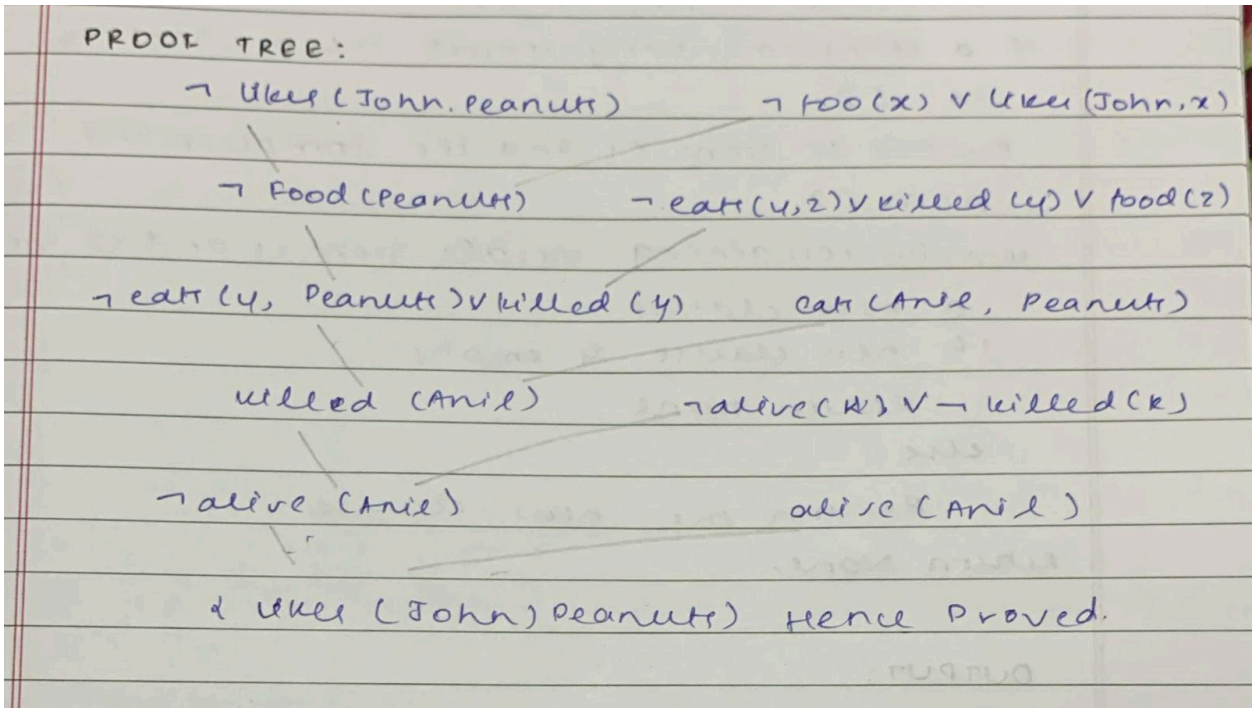
```
def resolve(clause1, clause2):
    resolved = set()
    for literal in clause1:
        if f"~{literal}" in clause2 or (literal.startswith("~") and literal[1:] in clause2):
            temp1 = clause1 - {literal}
            temp2 = clause2 - {f"~{literal}" if not literal.startswith("~") else literal[1:]}
            resolved = temp1.union(temp2)
            if not resolved:
                return None            return resolved
    return None

def resolution(kb, query):
    negated_query = {f"~{query}"}
    kb.append(negated_query)

    new_clauses = set()
    while True:
        pairs = combinations(kb, 2)
        for clause1, clause2 in pairs:
            resolvent = resolve(clause1, clause2)
```

```
            if resolvent is None:
                return True  # Query proven
            if resolvent:
                new_clauses.add(frozenset(resolvent))
        if new_clauses.issubset(set(kb)):
            return False  # No progress, query not proven
        kb.extend([set(clause) for clause in new_clauses])

# Define the KB in clausal form
kb = [
    {"~Food(x)", "Likes(John,x)"},  # John likes all food
    {"Food(Apple)"},                # Apple is food
    {"Food(Vegetables)"},           # Vegetables are food
    {"~Eats(x,y)", "Killed(x)", "Food(y)"},  # Eats and not killed -> food
    {"Eats(Anil,Peanuts)"},         # Anil eats peanuts
    {"~Killed(Anil)"},              # Anil is not killed
    {"~Eats(Anil,y)", "Eats(Harry,y)"},  # Anil eats -> Harry eats
]

query = "Likes(John,Peanuts)"
result = resolution(kb, query)

if result:
    print(f"The query '{query}' is PROVEN.")
else:
    print(f"The query '{query}' is NOT PROVEN.")
```

OUTPUT-

```
The query 'Likes(John,Peanuts)' is PROVEN.
```

PROOF TREE-


PROOF TREE:
¬ likes (John, Peanuts)
¬ Food (Peanuts)
¬ eats (y, Peanuts) V killed (y)
killed (Anil)
¬ alive (Anil)
¬ likes (John, Peanuts)
¬ Food (x) V likes (John, x)
¬ eats (y, z) V killed (y) V food (z)
eats (Anil, Peanuts)
¬ alive (k) V ¬ killed (k)
alive (Anil)
Hence Proved.