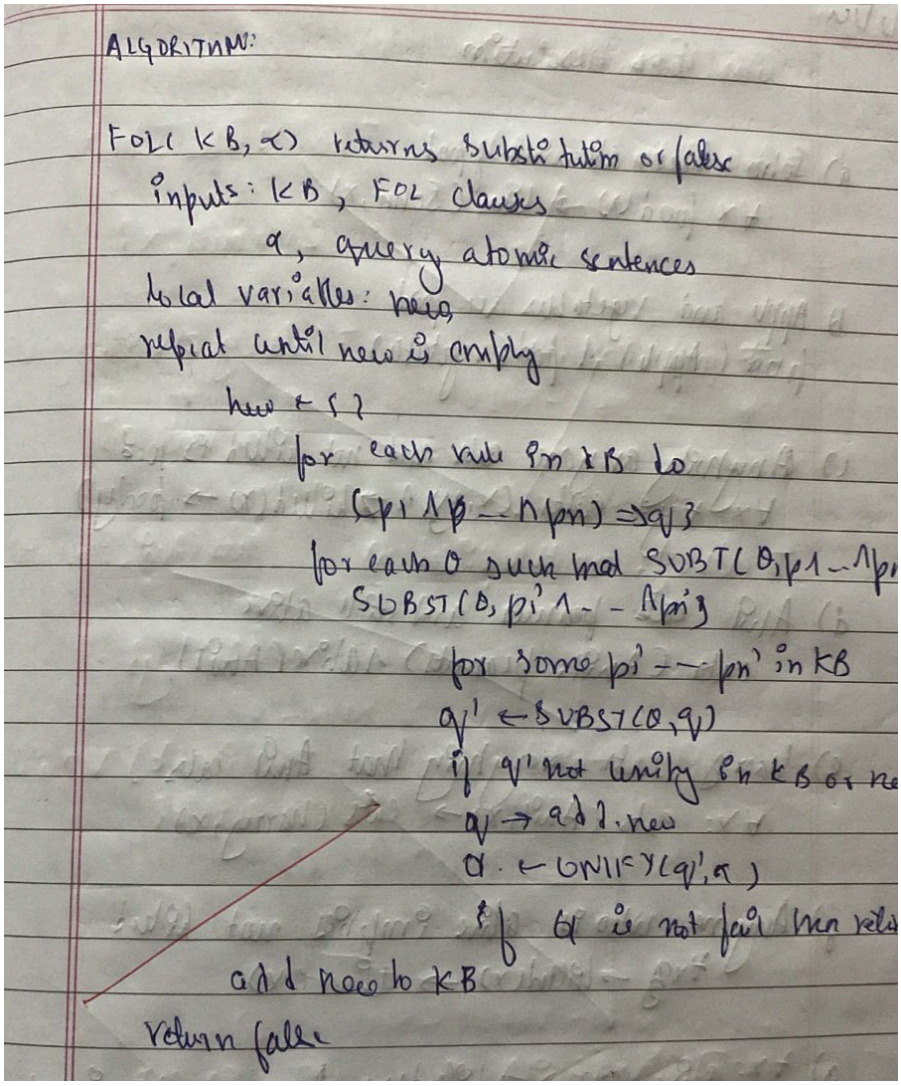
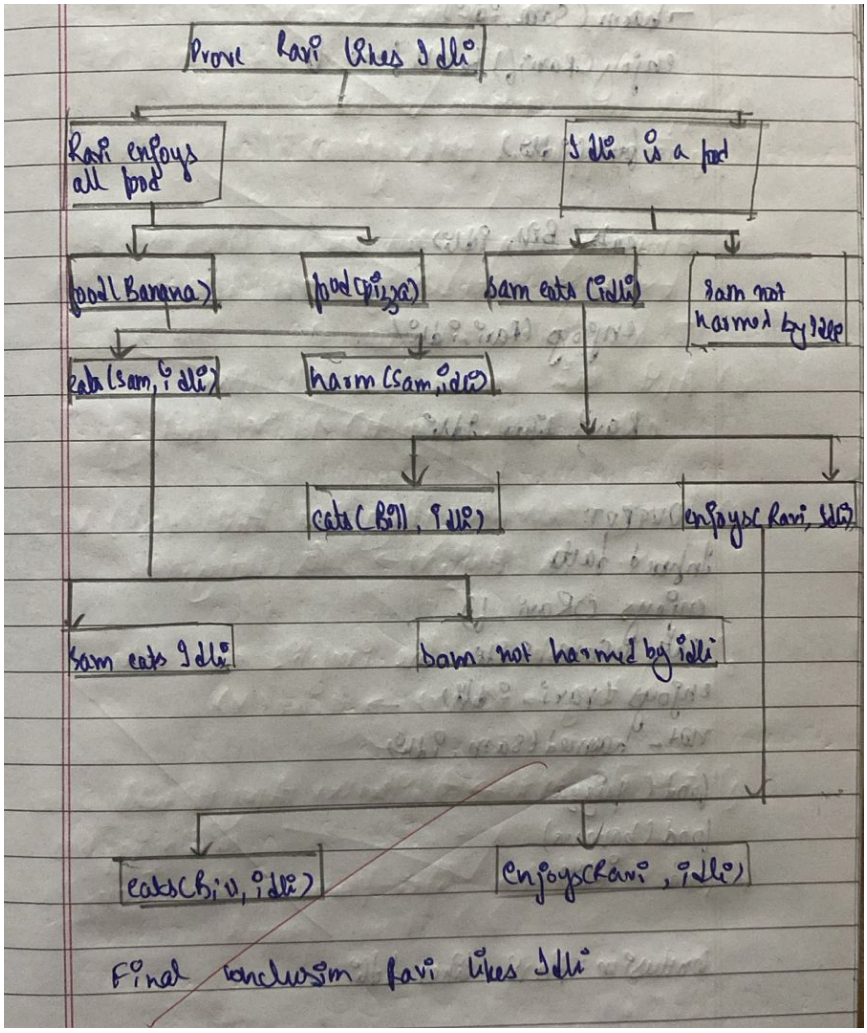
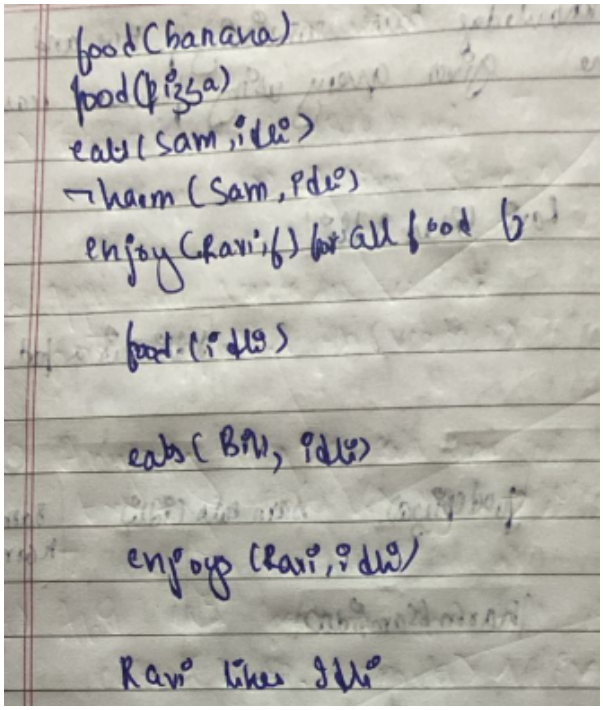
PROGRAM:12
Date:3/12/24
Program Title: Create a knowledge base consisting of first order logic statements and prove the given query using forward reasoning
Algorithm:



Tree:


Prove Ravi likes Idli
Ravi enjoys all food
Idli is a food
food(Banana)
food(pizza)
Sam eats Idli
Sam not harmed by Idli
eats(Sam, Idli)
harm(Sam, Idli)
eats(Bill, Idli)
enjoys(Ravi, Idli)
Sam eats Idli
Sam not harmed by Idli
eats(Bill, Idli)
enjoys(Ravi, Idli)
Final conclusion Ravi likes Idli



Code:

```
class KnowledgeBase:
    def __init__(self):
        self.facts = set()
        self.rules = []
    def add_fact(self, fact):
        self.facts.add(fact)
    def add_rule(self, rule):
        self.rules.append(rule)
    def forward_chain(self):
        new_facts = set(self.facts)
        while True:
            inferred = set()
            for rule in self.rules:
                conclusion, premises = rule
                if all(premise in self.facts for premise in premises):
                    if conclusion not in self.facts:
                        inferred.add(conclusion)
                        self.facts.add(conclusion)
            if not inferred:
                break
            new_facts.update(inferred)
        return new_facts
kb = KnowledgeBase()
kb.add_fact("food(banana)")
kb.add_fact("food(pizza)")
kb.add_fact("eats(Sam, idli)")
kb.add_fact("not_harmed(Sam, idli)")
kb.add_rule(("food(idli)", ["eats(Sam, idli)", "not_harmed(Sam, idli)"]))
kb.add_rule(("eats(Bill, idli)", ["eats(Sam, idli)"]))
kb.add_rule(("enjoys(Ravi, idli)", ["food(idli)"]))
inferred_facts = kb.forward_chain()
print("Inferred Facts:")
for fact in inferred_facts:
    print(fact)
if "enjoys(Ravi, idli)" in inferred_facts:
    print("\nConclusion: Ravi likes Idli.")
else:
    print("\nConclusion: Ravi does not like Idli.")
```

class KnowledgeBase:

```
    def __init__(self):
        self.facts = set()
        self.rules = []
    def add_fact(self, fact):
        self.facts.add(fact)
    def add_rule(self, rule):
        self.rules.append(rule)
    def forward_chain(self):
        new_facts = set(self.facts)
        while True:
            inferred = set()
            for rule in self.rules:
                conclusion, premises = rule
                if all(premise in self.facts for premise in premises):
                    if conclusion not in self.facts:
                        inferred.add(conclusion)
                        self.facts.add(conclusion)
            if not inferred:
                break
            new_facts.update(inferred)
        return new_facts
kb = KnowledgeBase()
kb.add_fact("food(banana)")
kb.add_fact("food(pizza)")
kb.add_fact("eats(Sam, idli)")
kb.add_fact("not_harmed(Sam, idli)")
kb.add_rule(("food(idli)", ["eats(Sam, idli)", "not_harmed(Sam, idli)"]))
kb.add_rule(("eats(Bill, idli)", ["eats(Sam, idli)"]))
kb.add_rule(("enjoys(Ravi, idli)", ["food(idli)"]))
inferred_facts = kb.forward_chain()
print("Inferred Facts:")
for fact in inferred_facts:
    print(fact)
if "enjoys(Ravi, idli)" in inferred_facts:
    print("\nConclusion: Ravi likes Idli.")
else:
    print("\nConclusion: Ravi does not like Idli.")
```

Snapshot of the output:

```
Inferred Facts:
food(pizza)
not_harmed(Sam, idli)
eats(Sam, idli)
eats(Bill, idli)
enjoys(Ravi, idli)
food(idli)
food(banana)

Conclusion: Ravi likes Idli.
```