

## 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:

**ALGORITHM:**

For  $\text{KB}, \alpha$  returns substitution or false

Inputs:  $\text{KB}$ , FOL clauses  
 $\alpha$ , query atomic sentences  
local variables: new

repeat until new is empty

    new ← {}

    for each rule  $\beta \rightarrow \gamma$  in  $\text{KB}$  do

$(\beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n) \Rightarrow \gamma$

        for each  $\theta$  such that  $\text{SUBT}(\theta, \beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n)$   
 $\text{SUBST}(\theta, \beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n)$

            for some  $\beta'_1 \wedge \beta'_2 \wedge \dots \wedge \beta'_n$  in  $\text{KB}$

$\alpha' \leftarrow \text{SUBST}(\theta, \alpha)$

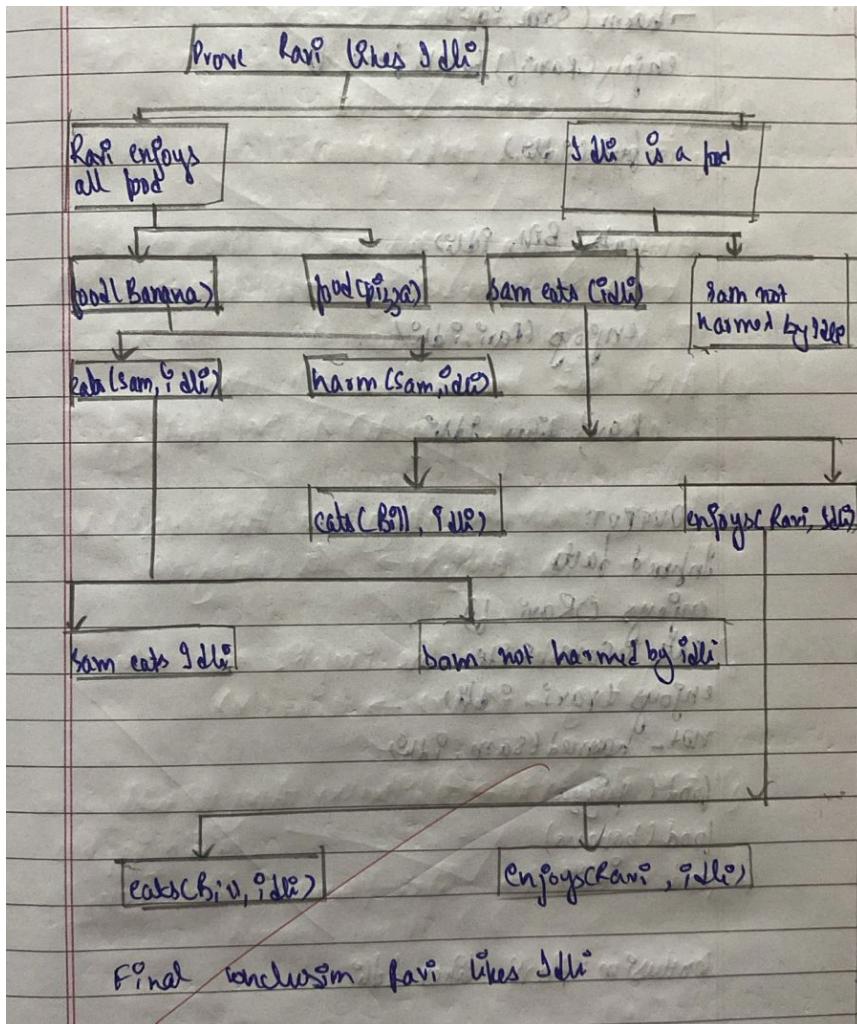
                if  $\alpha'$  not unify  $\beta_n$  in  $\text{KB}$  or not  
 $\alpha' \rightarrow \alpha$  add to new

$\alpha \leftarrow \text{UNIFY}(\alpha', \alpha)$

                if  $\alpha$  is not fail then return  
and new to  $\text{KB}$

    return false

Tree:



food(banana)  
 food(pizza)  
 eats(Sam, idli)  
 $\neg$ harmed(Sam, Pde)  
 enjoys(Ravi, f) for all food f  
 food(idli)  
 eats(Bill, idli)  
 enjoys(Ravi, idli)  
 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.

OUTPUT:  
Inferred facts  
enjoys(Ravi, idli)  
eats(Sam, idli)  
enjoys(Ravi, idli)  
not\_harmed(Sam, idli)  
food(pizza)  
food(banana)  
food(idli)  
Inferred facts