Forward Chaining

class ForwardChaining:

def \_\_init\_\_(self):

self.facts = set() # Known facts

self.rules = [] # Inference rules

def add\_fact(self, fact):

self.facts.add(fact)

def add\_rule(self, conditions, conclusion):

self.rules.append((conditions, conclusion))

def forward\_chain(self):

new\_facts = set(self.facts)

while True:

applied\_rule = False

for conditions, conclusion in self.rules:

# If the conditions are in the known facts and the conclusion isn't already known, apply the #rule

if conditions.issubset(new\_facts) and conclusion not in new\_facts:

new\_facts.add(conclusion)

applied\_rule = True

if not applied\_rule:

break

return new\_facts

def get\_input\_forward():

fc = ForwardChaining()

n = int(input("Enter number of facts: "))

for \_ in range(n):

fact = input("Enter fact: ")

fc.add\_fact(fact)

m = int(input("Enter number of rules: "))

for \_ in range(m):

rule\_input = input("Enter rule in format 'if fact1, fact2 then conclusion': ").split(" then ")

conditions = set(rule\_input[0].split(", "))

conclusion = rule\_input[1].strip()

fc.add\_rule(conditions, conclusion)

new\_facts = fc.forward\_chain()

print("Derived facts:", new\_facts)

# Main driver

get\_input\_forward()

OUTPUT:

Enter number of facts: 2

Enter fact: A

Enter fact: B

Enter number of rules: 2

Enter rule in format 'if fact1, fact2 then conclusion': A, B then C

Enter rule in format 'if fact1, fact2 then conclusion': C then D

Derived facts: {'A', 'B', 'C', 'D'}