# **Backward Chaining Algorithm**

### **Aim:**

### To implement the backward chaining algorithm using Python, in order to determine if a goal can be proven based on a given knowledge base (set of rules and known facts).

### **Procedure:**

1. Define a knowledge base using rules, where each rule's key depends on the list of conditions (subgoals).
2. Specify known facts that are initially assumed to be true.
3. Define a recursive backward\_chain function that:
   1. Returns True if the goal is in known facts.
   2. Tries to prove all subgoals recursively if the goal has rules.
   3. Adds the goal to facts if all subgoals are proven.
4. Prompt the user to enter a goal.
5. Use the backward\_chain function to determine if the goal can be proven.
6. Display the result accordingly.

### **Program:**

rules = {

"C": ["A", "B"],

"E": ["C", "D"],

"F": ["E"]

}

facts = {"A", "B", "D"}

def backward\_chain(goal):

if goal in facts:

return True

if goal not in rules:

return False

for subgoal in rules[goal]:

if not backward\_chain(subgoal):

return False

facts.add(goal) # Mark it as proven

return True

goal = input("Enter the goal to prove (e.g., E): ").strip().upper()

if backward\_chain(goal):

print(f"YES, '{goal}' can be proven.")

else:

print(f"NO, '{goal}' cannot be proven with current knowledge.")

### **Input:**

Enter the goal to prove (e.g., E): E

### **Output:**

YES, 'E' can be proven.

**Result:**

The program correctly applies backward chaining logic to check if the goal can be derived from the rules and known facts. It recursively verifies subgoals and updates the knowledge base accordingly.