LU18- Forward Chaining

LU Objectives	
To explain forward chaining algorithm	
LU Outcomes	CO : 3
Apply forward chaining algorithm to solve problems	

Forward Chaining

Forward chaining:-

⇒ It is a form of reasoning which starts with atomic sentences in the knowledge base and applies inference rules in the forward direction to extract more data until a goal is reached.

Properties :-

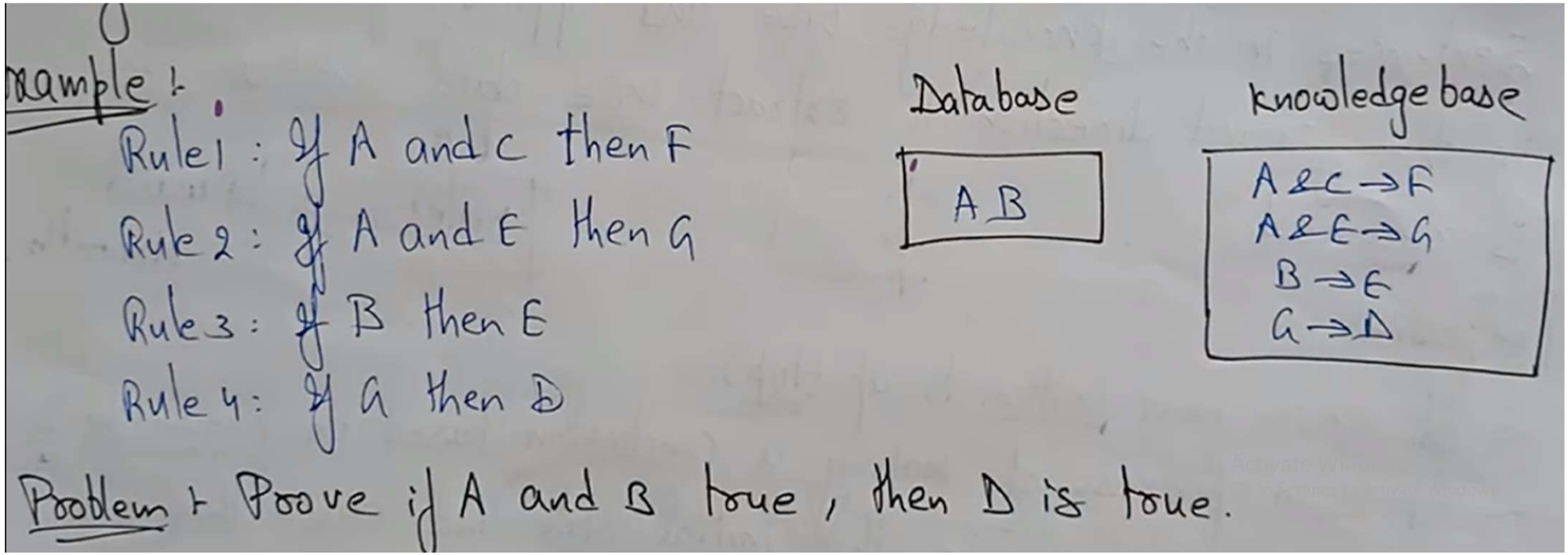
→ It moves from bottom to up (top)

→ It is a process of making a conclusion based on known facts of data, by starting from the initial state and reach the goal state

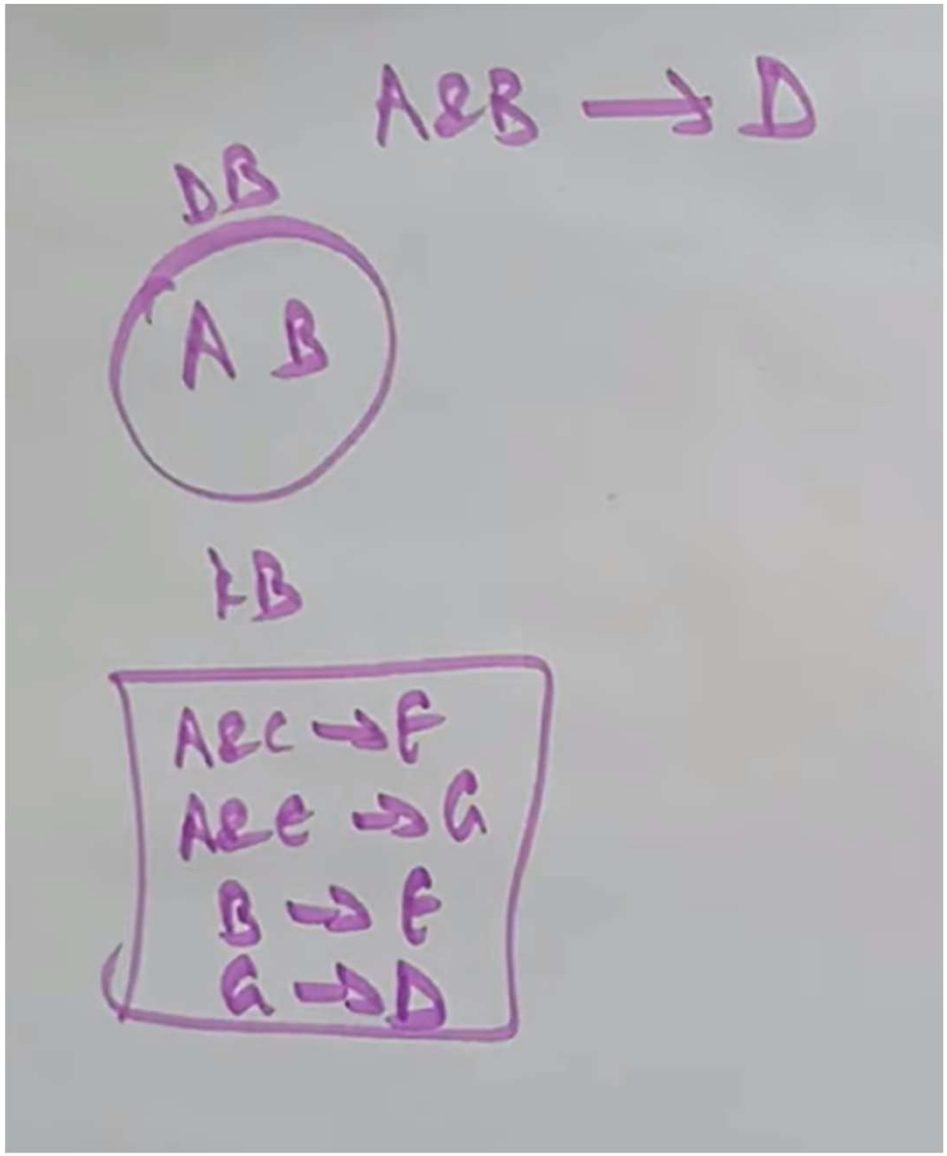
→ Forward chaining approach is also called as data-driven as we reach to the goal using available data.

→ Forward - chaining approach is commonly used in the expert system.

Forward Chaining

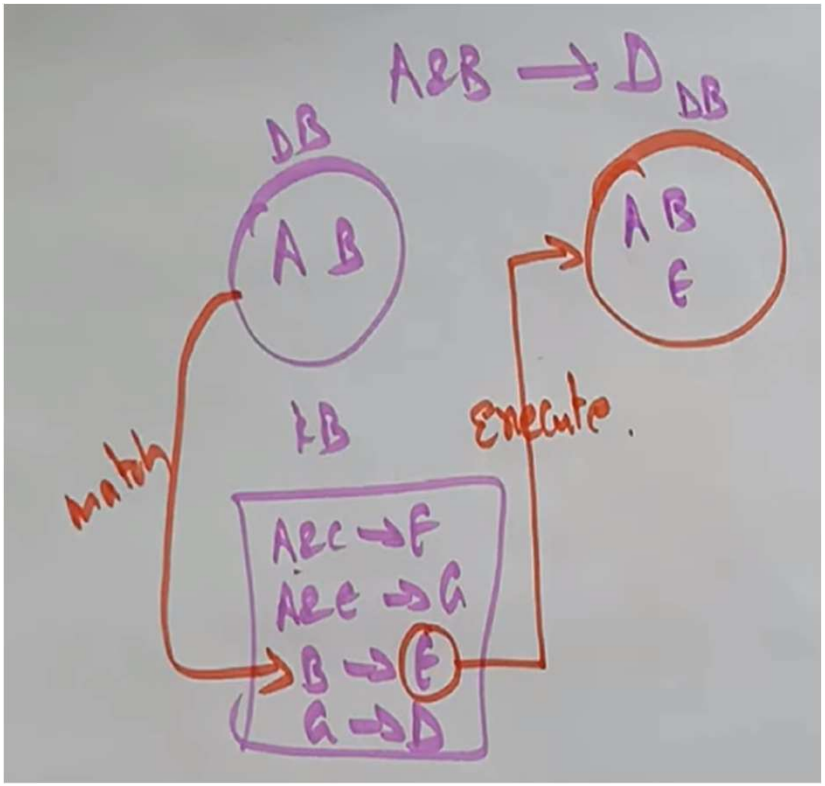


Forward Chaining



- A and B are true in the Database(DB)
- Need to prove: D should also be true and added to the DB
- Rules are present in KB
- Need to match

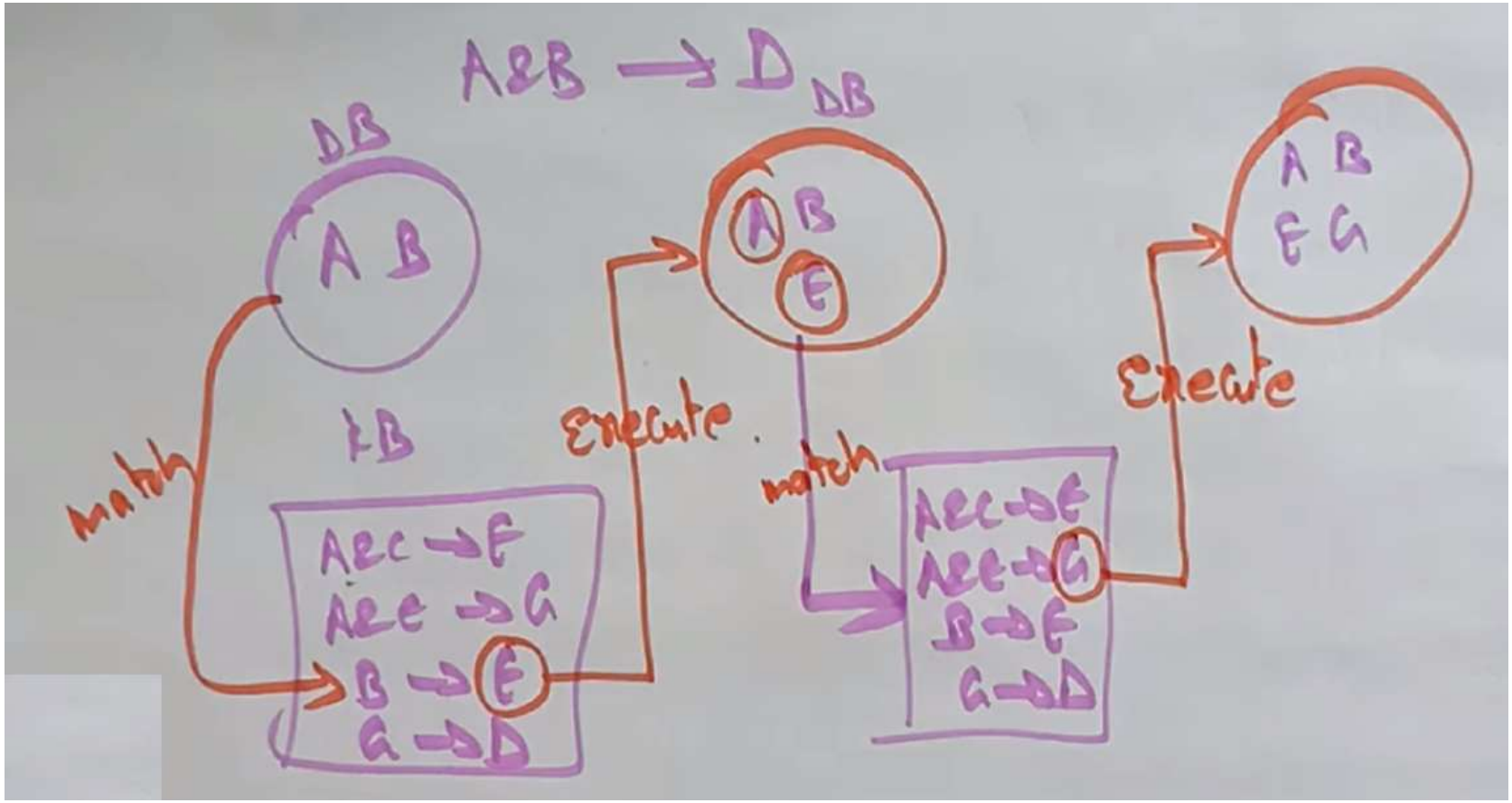
Forward Chaining



- **<u>Need to match</u>**
- Inside DB, A & B combination is there
- Whereas A & B combination or simple A is not there in Knowledge Base(KB)
- Since, only B is in KB, match it with the KB
- Output of B is ? B-> E means
  - Place(**Execute**) E on DB

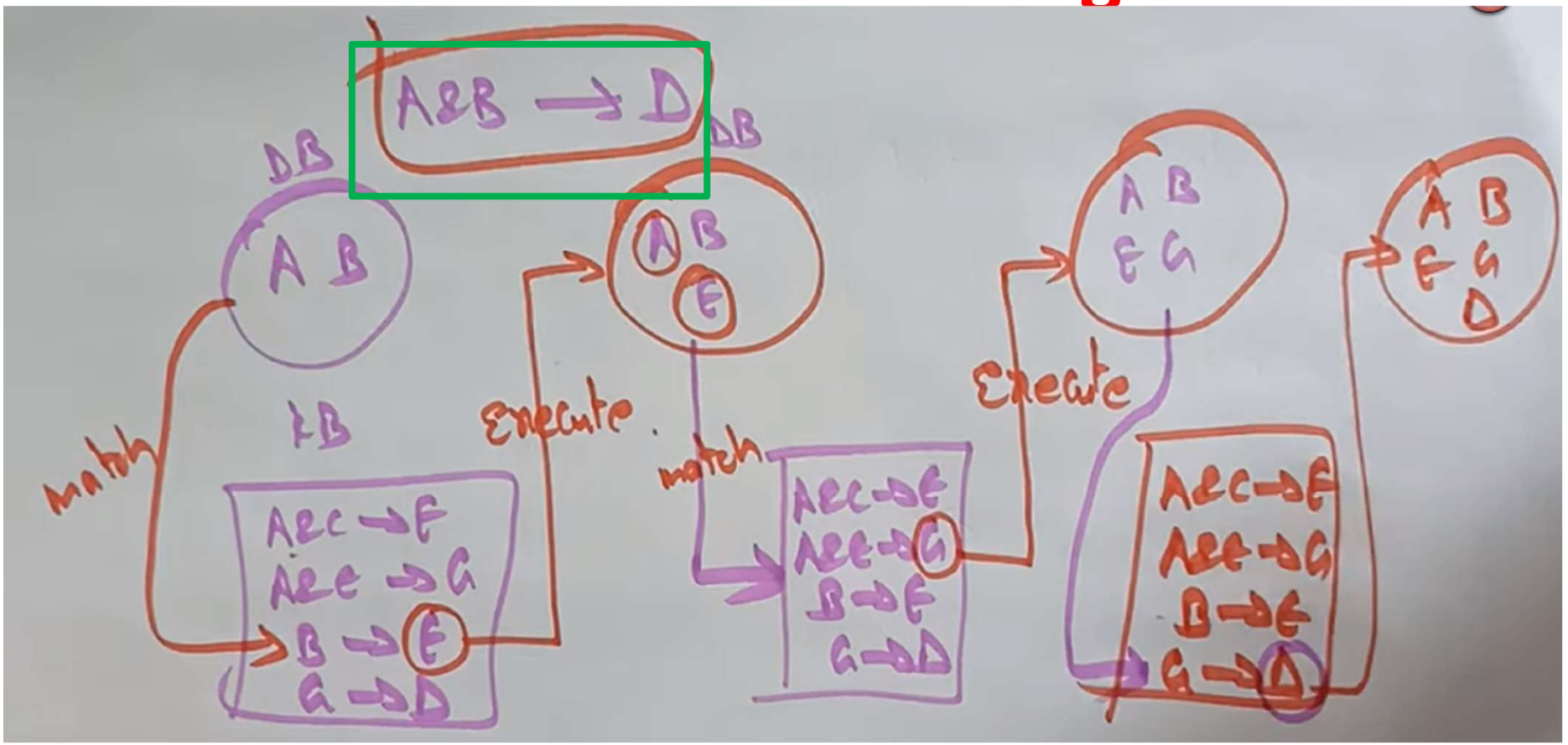
- **Next Repeat the process between DB (current) and KB**

Forward Chaining



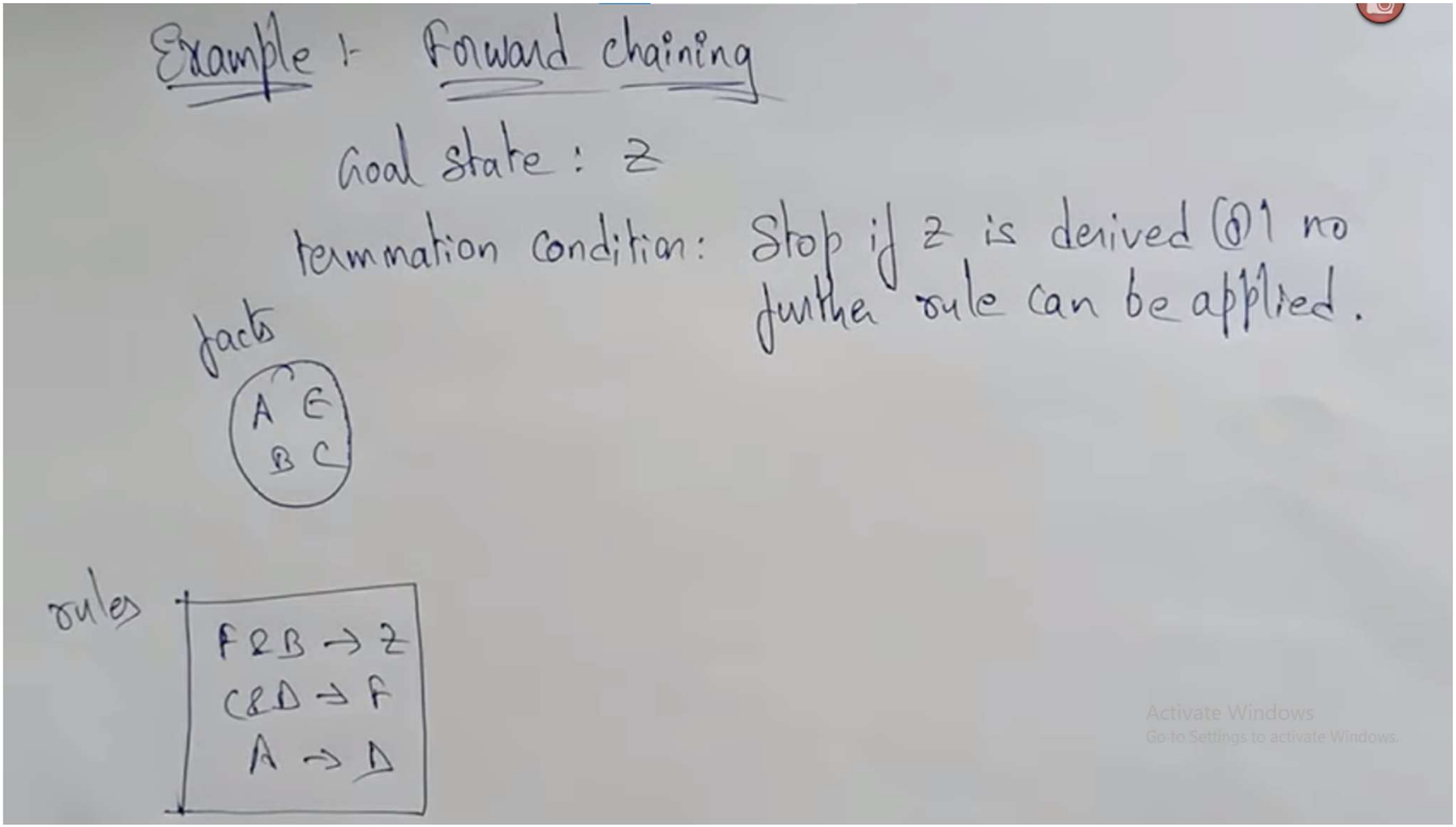
- A & E combination is in KB
- So Execute G in DB

Forward Chaining

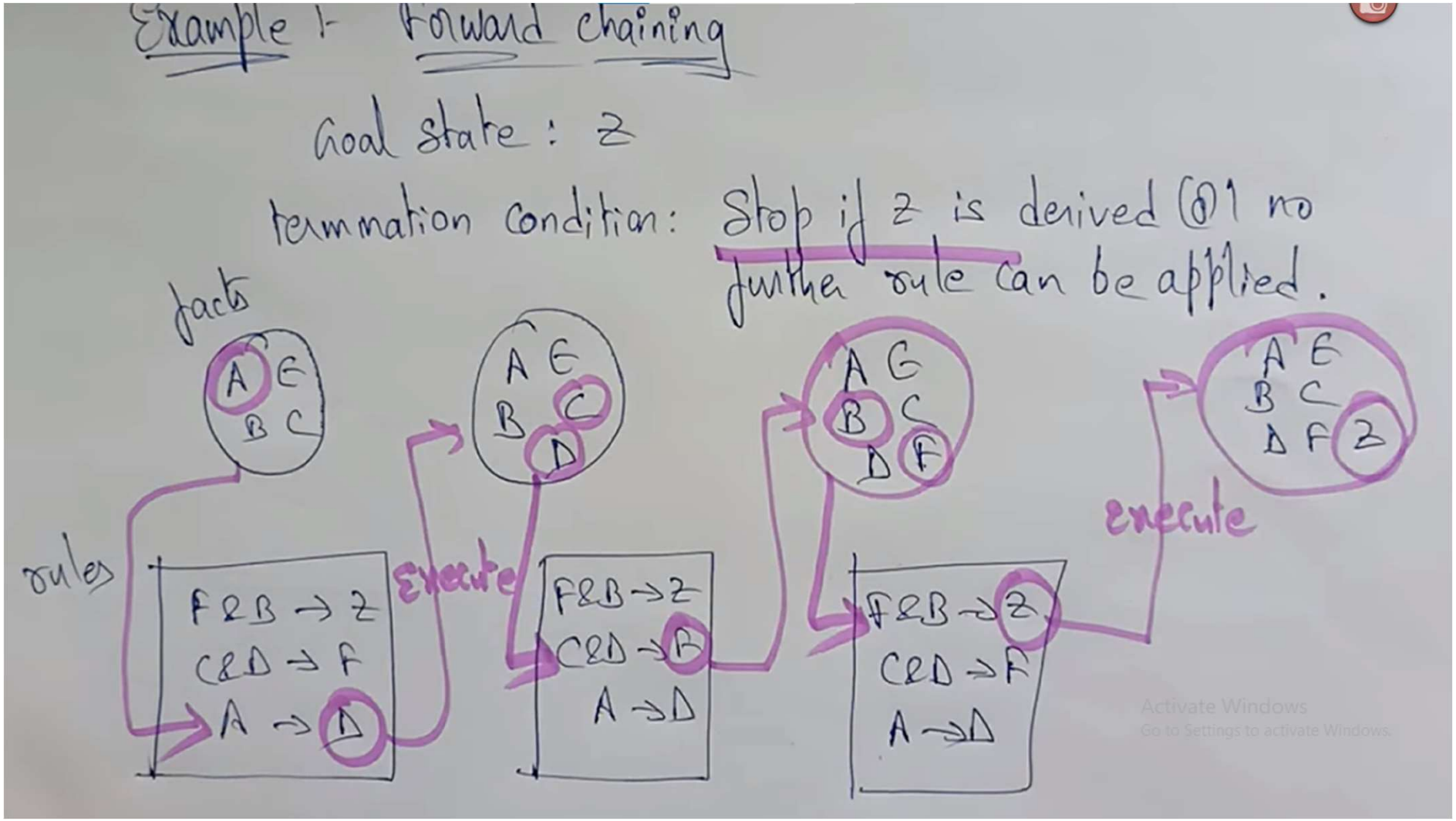


- A or A & B or E is not there in KB (it is not matching)
- **Single G is there (means G is matching with an item KB)**
- **Execute D in DB**
- **Now DB contains A, B, E, G and D**
- **Reach Goal ? Yes [means whwnever A & B are executed then D is also executed]**

Forward Chaining – Example – Try on yourself



Forward Chaining – Example – Solution



Forward Chaining

- The idea is simple: start with the atomic sentences in the knowledge base
- Now apply Modus Ponens in the forward direction, adding new atomic sentences, until no further inferences can be made.

First-order definite clauses

- First-order definite clauses closely resemble propositional definite clauses
- They are disjunctions of literals of which *exactly one is positive.*
- *A definite clause either is atomic* or is an implication whose antecedent is a conjunction of positive literals and whose consequent is a single positive literal.

First-order definite clauses

- EX:

  King(x) ∧ Greedy(x) ⇒ Evil(x) .

  King(John) .

  Greedy(y)

- Unlike propositional literals, first-order literals can include variables, in which case those variables are assumed to be universally quantified. **(it is omitted).**
- Not every knowledge base can be converted into a set of definite clauses because of the single-positive-literal restriction, but many can.

First-order definite clauses

- Consider the sentence
- “The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.”
- Let us prove **“West is a criminal”.**
- First, we will represent these facts as first-order definite
- clauses.

1)“. . . it is a crime for an American to sell weapons to hostile nations”:

American(x) ∧Weapon(y) ∧ Sells(x, y, z) ∧ Hostile(z) ⇒ Criminal (x) .

First-order definite clauses

2) “Nono . . . has some missiles.” The sentence $\exists$ x Owns(Nono, x)$\wedge$Missile(x) is transformed into two definite clauses by Existential Instantiation, introducing a new constant M1:

Owns(Nono,M1)

Missile(M1)

3) “All of its missiles were sold to it by Colonel West”:

$\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$.

4) We will also need to know that missiles are weapons:

$\text{Missile}(x) \Rightarrow \text{Weapon}(x)$

First-order definite clauses

5) we must know that an enemy of America counts as "hostile":

$Enemy(x,America) \Rightarrow Hostile(x)$

6) "West, who is American . . .":

$American(West)$

7) "The country Nono, an enemy of America . . .":

$Enemy(Nono,America)$

- **Datalog** is a language that is restricted to first-order definite clauses with no function symbols.

Forward chaining algorithm

function FOL-FC-ASK(KB, α) **returns** a substitution or *false*

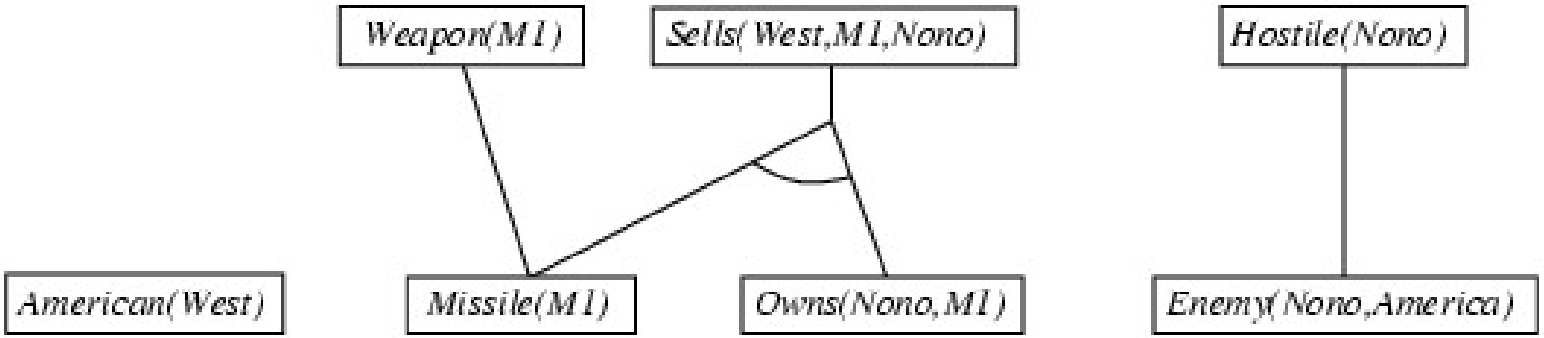
repeat until *new* is empty
  new $\leftarrow \{\}$
  for each sentence r **in** KB **do**
    $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART(r)
    for each θ such that $(p_1 \wedge \ldots \wedge p_n)\theta = (p'_1 \wedge \ldots \wedge p'_n)\theta$
        for some $p'_1, \ldots, p'_n$ in KB
      $q' \leftarrow$ SUBST(θ, q)
      if q' is not a renaming of a sentence already in KB or *new* **then do**
        add q' to *new*
        $\phi \leftarrow$ UNIFY(q', α)
        if ϕ is not *fail* **then return** ϕ
  add *new* to KB
return *false*

Explanation of FC

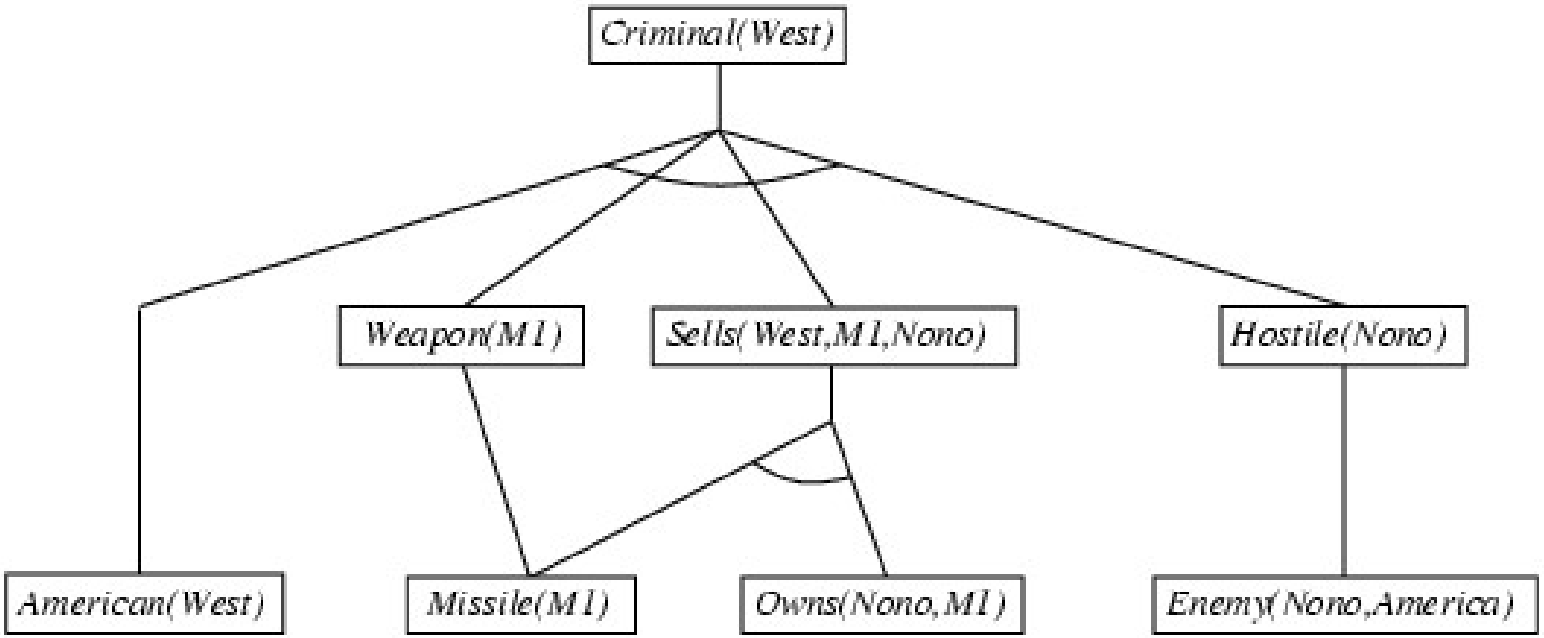
- The above example requires two iterations.
- On the first iteration, rule (1) has unsatisfied premises.
- Rule (3) is satisfied with {x/M1}, and Sells(West,M1, Nono) is added.
- Rule (4) is satisfied with {x/M1}, and Weapon(M1) is added.
- Rule (5) is satisfied with {x/Nono}, and Hostile(Nono) is added.
- • On the second iteration, rule (1) is satisfied with {x/West, y/M1, z/Nono}, and Criminal (West) is added

Forward chaining Proof Tree

Forward chaining proof



Forward chaining proof



Properties of forward chaining

- Sound and complete for first-order definite clauses
-
- Datalog = first-order definite clauses + no functions
- FC terminates for Datalog in finite number of iterations
-
- May not terminate in general if α is not entailed
-
- This is unavoidable: entailment with definite clauses is semidecidable
-

Efficiency of forward chaining

Incremental forward chaining: no need to match a rule on iteration *k* if a premise wasn't added on iteration *k-1*

$\Rightarrow$ match each rule whose premise contains a newly added positive literal

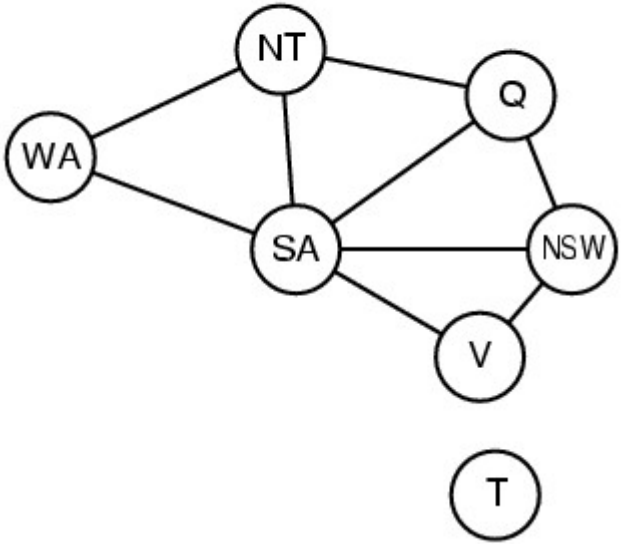
Matching itself can be expensive:

Database indexing allows O(1) retrieval of known facts

- e.g., query *Missile(x)* retrieves *Missile(M_1)*
-

Forward chaining is widely used in deductive databases

Hard matching example



$Diff(wa,nt) \wedge Diff(wa,sa) \wedge Diff(nt,q) \wedge Diff(nt,sa) \wedge Diff(q,nsw) \wedge Diff(q,sa) \wedge Diff(nsw,v) \wedge Diff(nsw,sa) \wedge Diff(v,sa) \Rightarrow Colorable()$

$Diff(Red,Blue)$ $Diff(Red,Green)$
$Diff(Green,Red)$ $Diff(Green,Blue)$
$Diff(Blue,Red)$ $Diff(Blue,Green)$

- *Colorable*() is inferred iff the CSP has a solution
- CSPs include 3SAT as a special case, hence matching is NP-hard
-