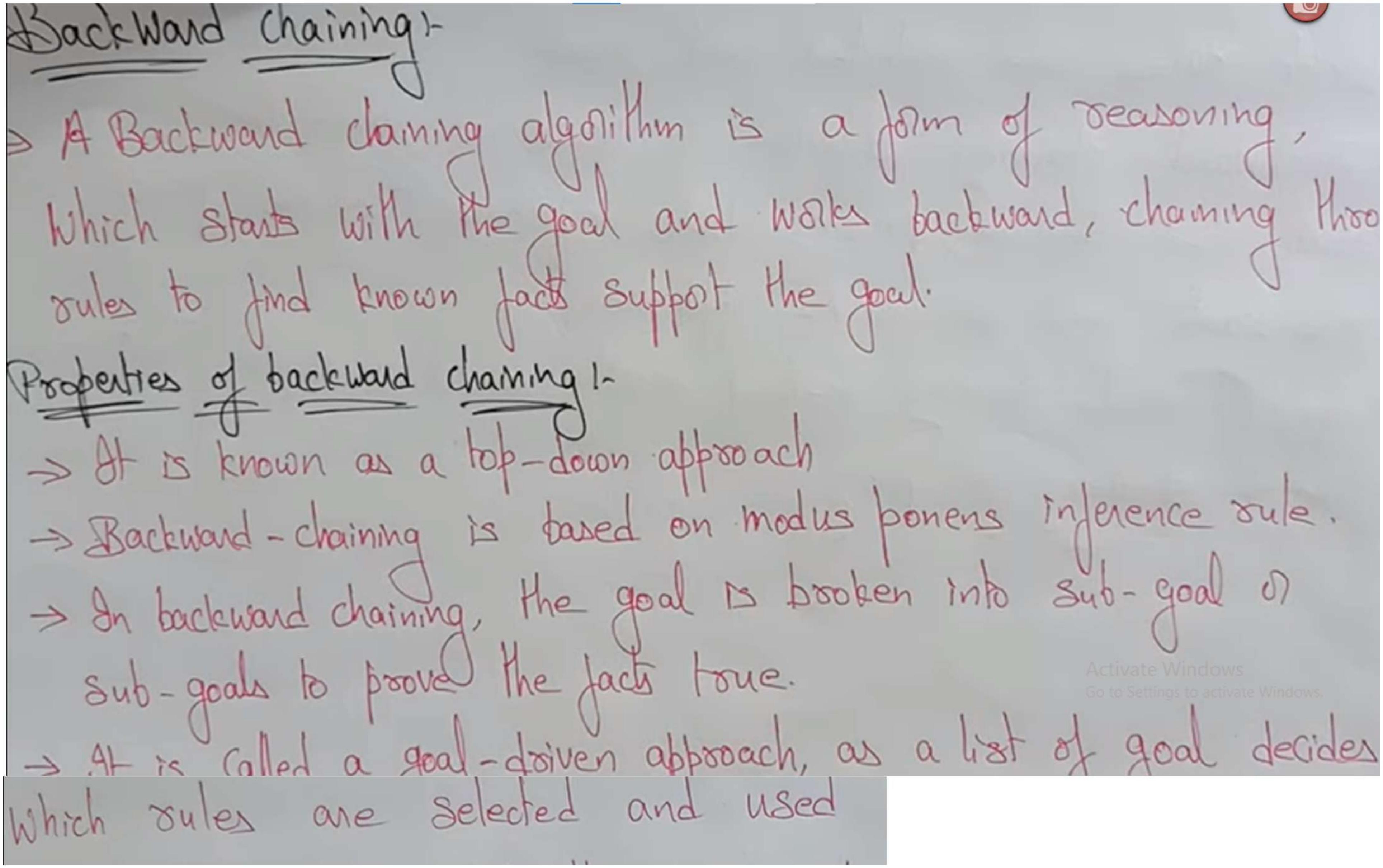
LU-19: Backward Chaining

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

Backward Chaining



Backward Chaining:-

→ A Backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining thro rules to find known facts support the goal.

Properties of backward chaining:-

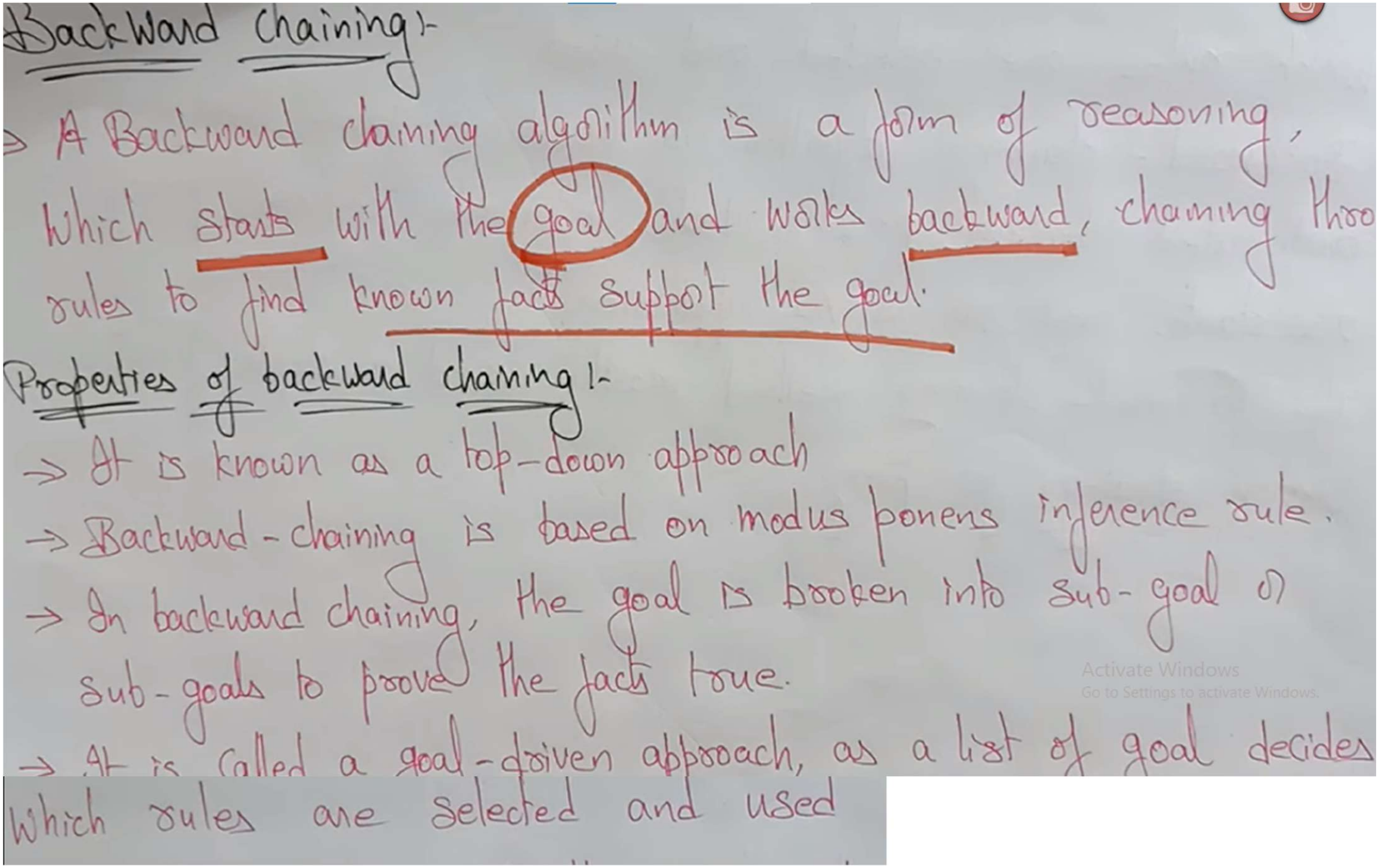
→ It is known as a top-down approach

→ Backward-chaining is based on modus ponens inference rule.

→ In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.

→ It is called a goal-driven approach, as a list of goal decides which rules are selected and used

Backward Chaining



Backward chaining:-

→ A Backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining thro rules to find known facts support the goal.

Properties of backward chaining:-

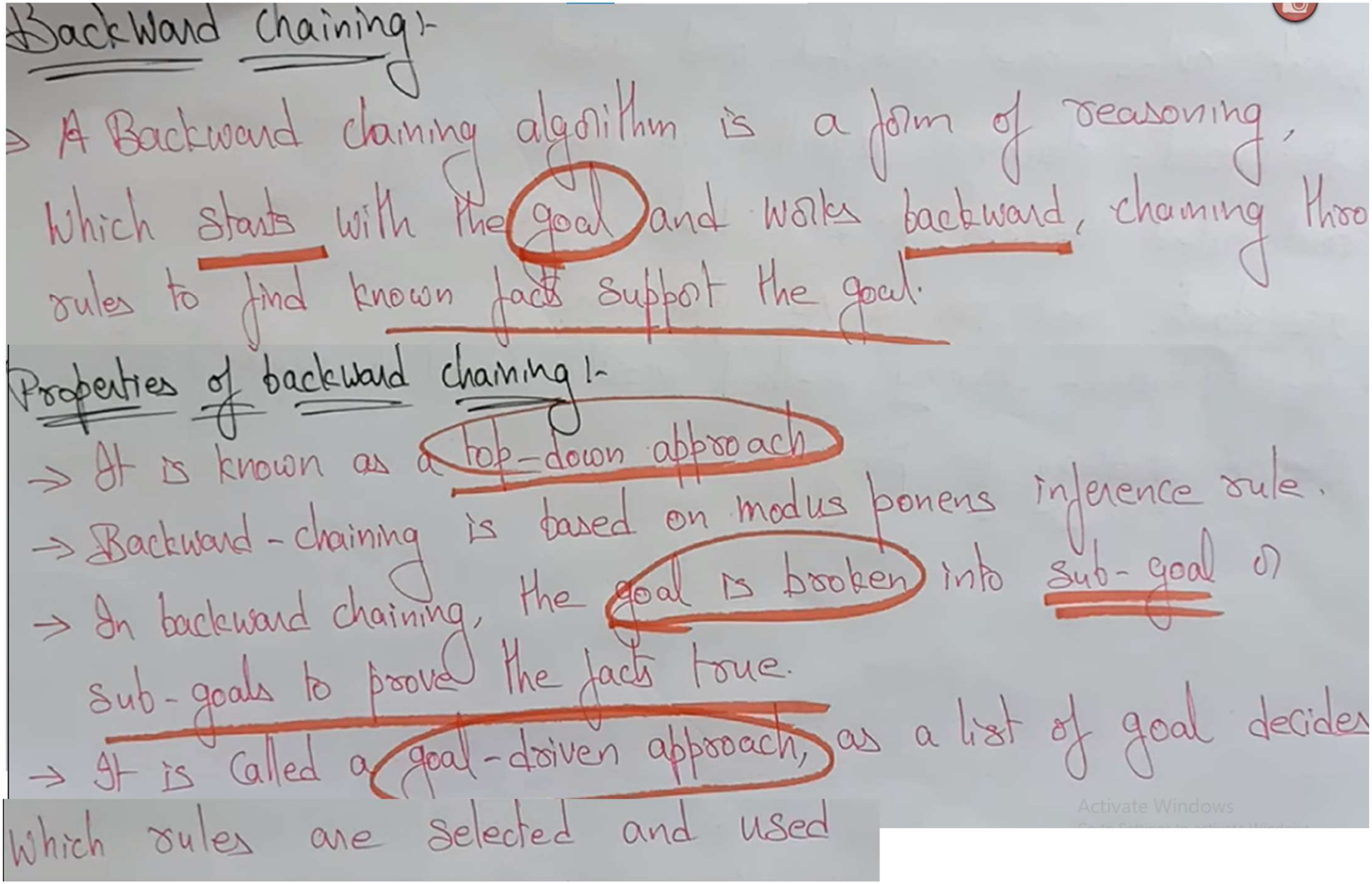
→ It is known as a top-down approach

→ Backward-chaining is based on modus ponens inference rule.

→ In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.

→ It is called a goal-driven approach, as a list of goal decides which rules are selected and used

Backward Chaining



Backward chaining:-

→ A Backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining thro rules to find known facts support the goal.

Properties of backward chaining:-

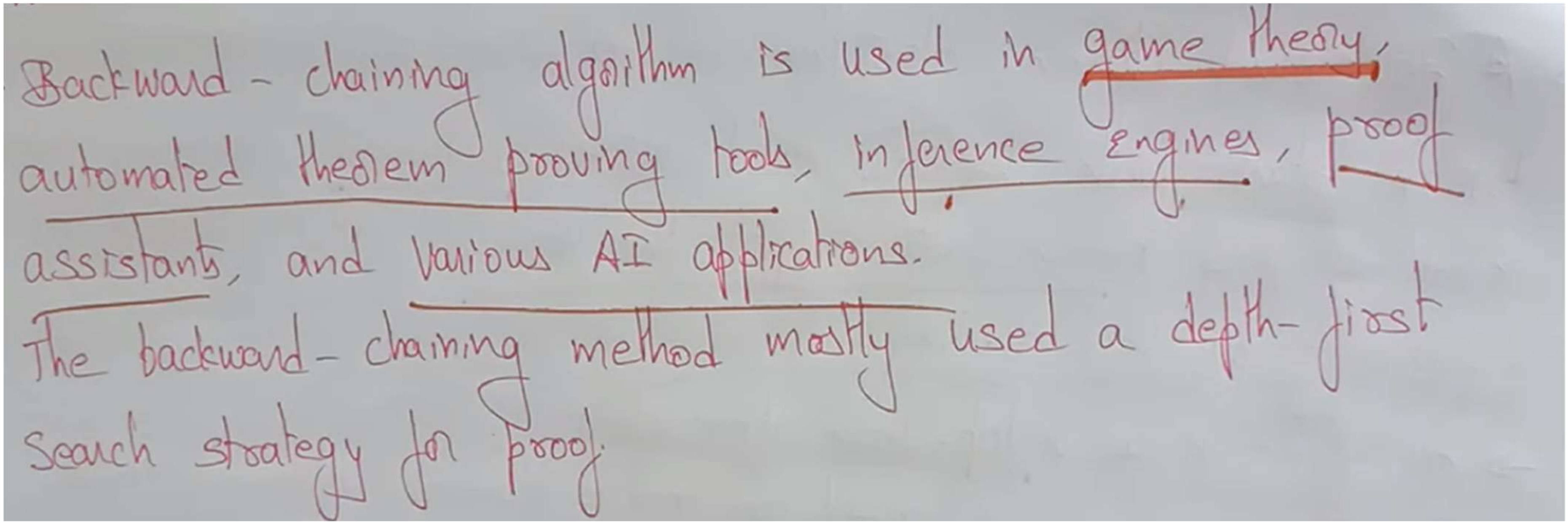
→ It is known as a top-down approach

→ Backward-chaining is based on modus ponens inference rule.

→ In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.

→ It is called a goal-driven approach, as a list of goal decides which rules are selected and used

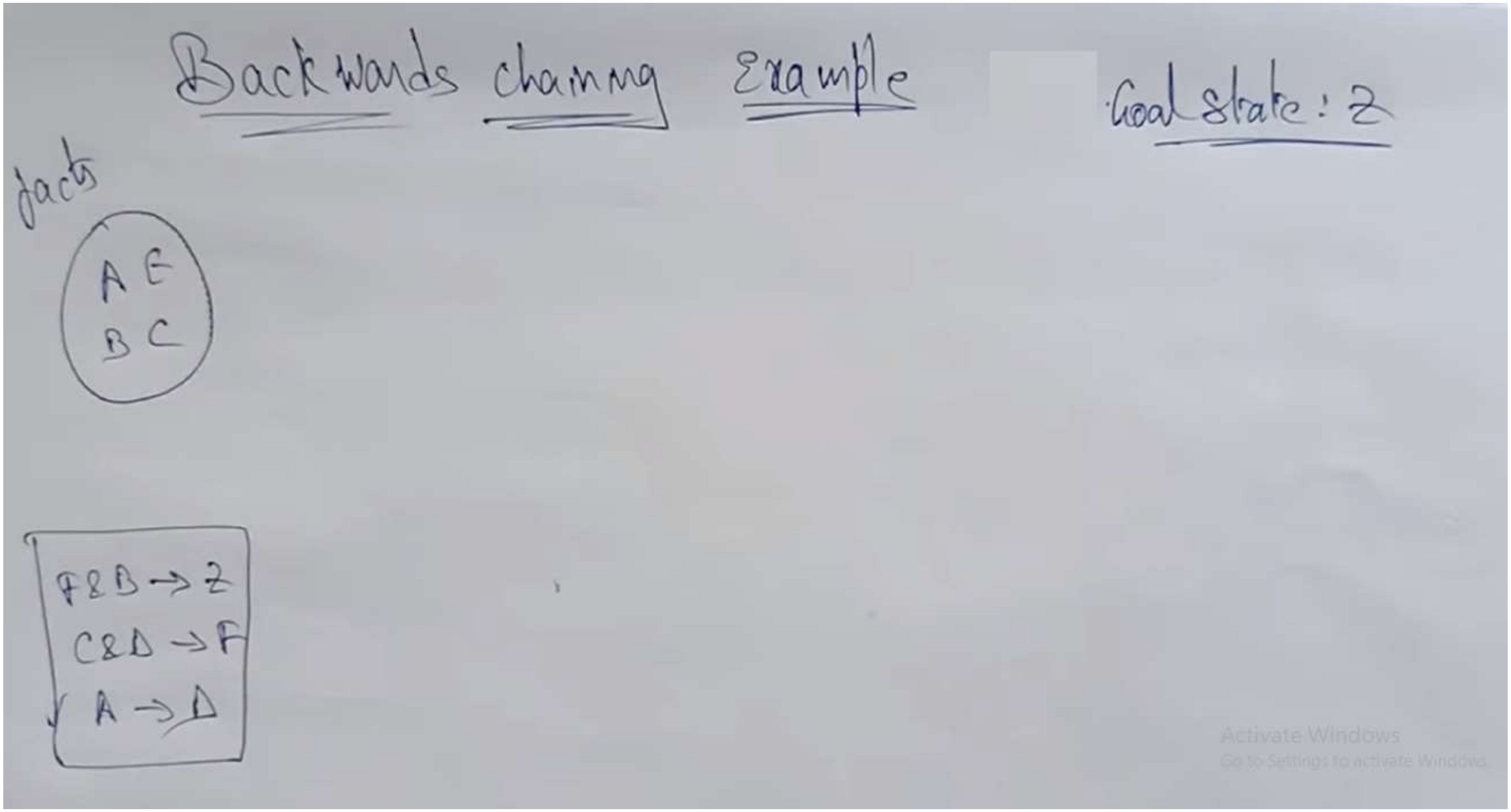
Backward Chaining



Backward - chaining algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.

The backward - chaining method mostly used a depth-first search strategy for proof.

Backward Chaining



Backwards chaining Example

Goal state : Z

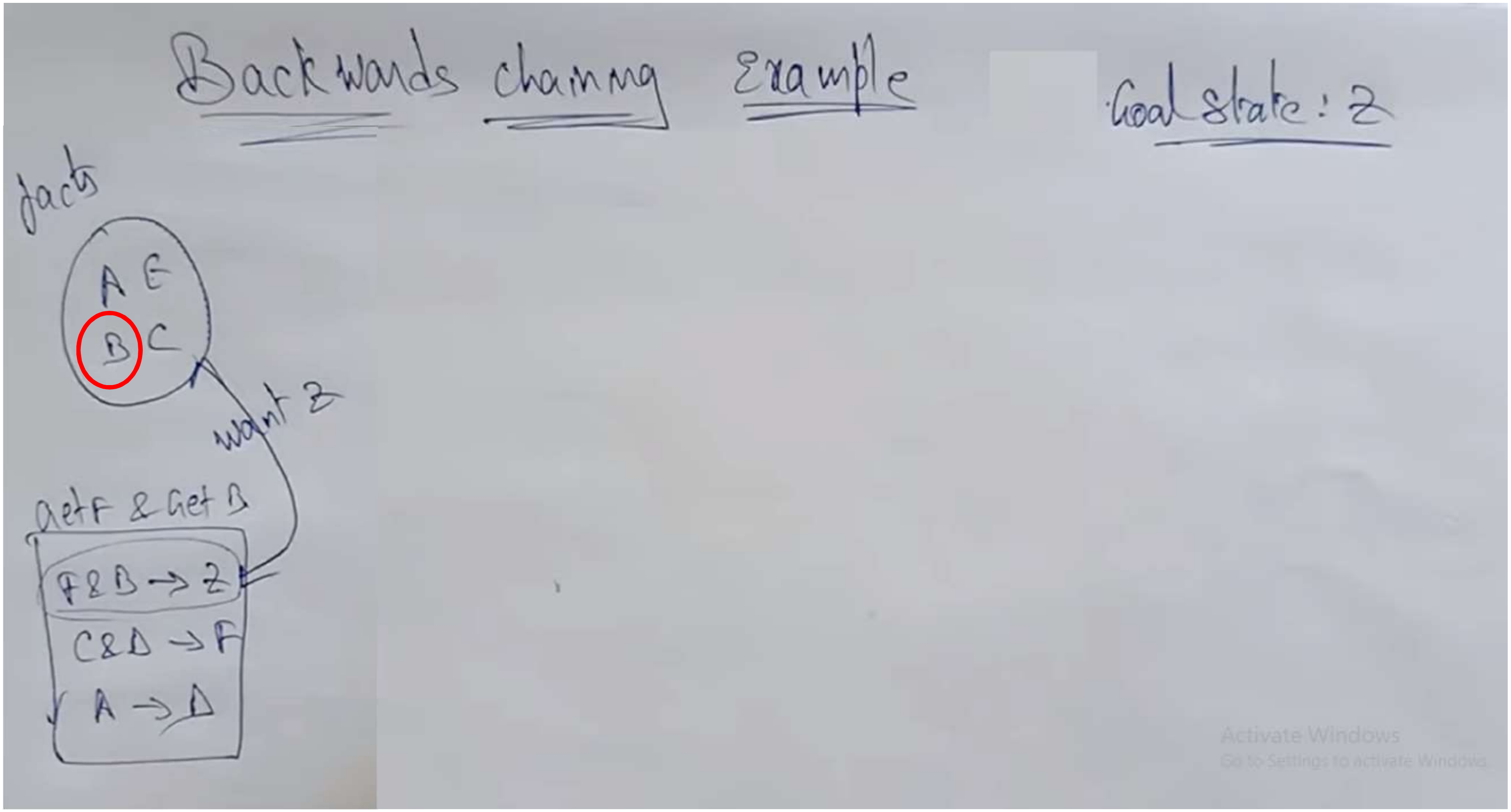
facts

A E
B C

F&B → Z
C&D → F
A → D

In the Backward: Want Z which is Goal state
Where is Z ?
Z is present in the Rule F & B -> Z . So Get F & Get B

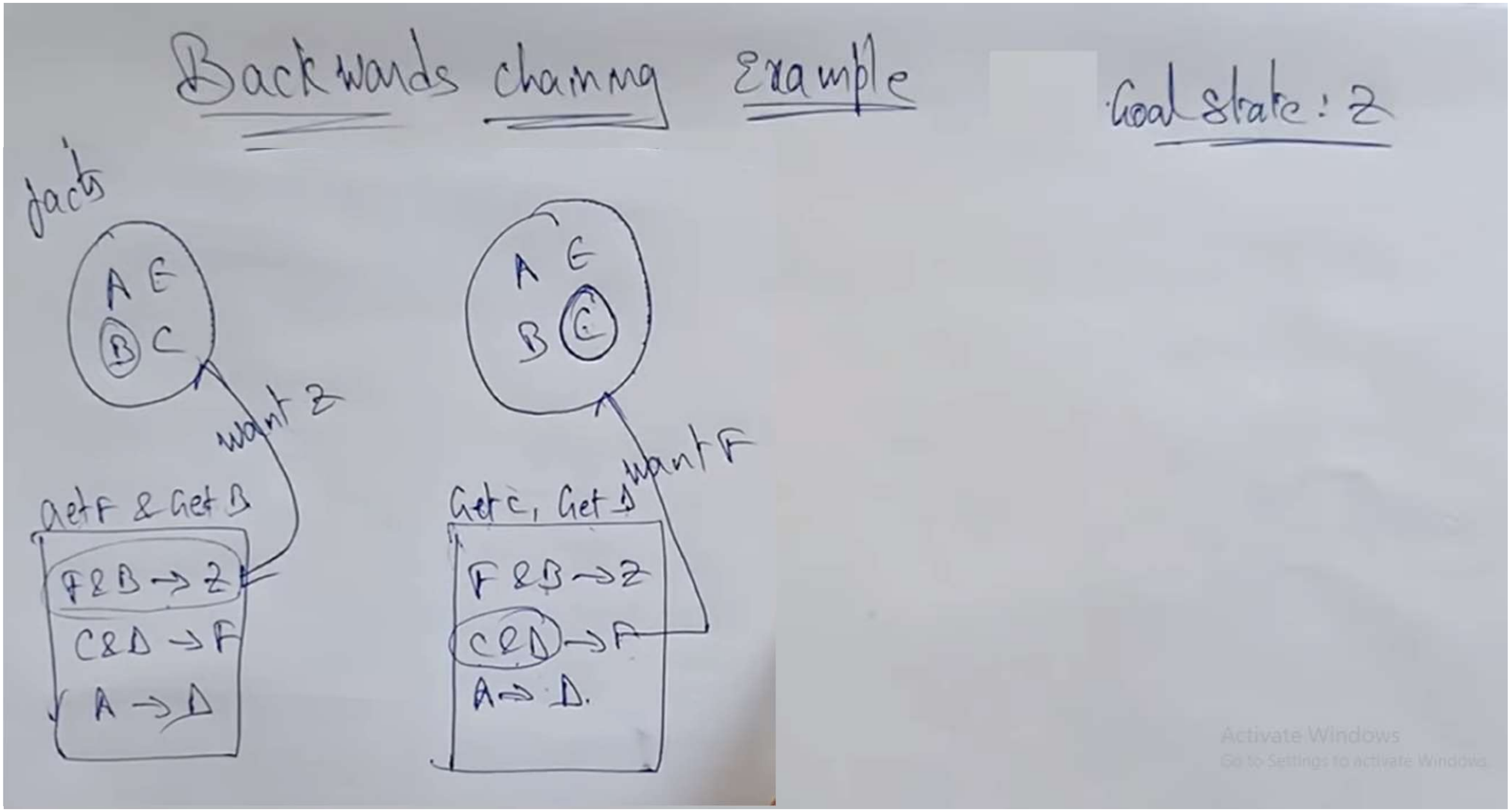
Backward Chaining



Only B in KB is matching here.

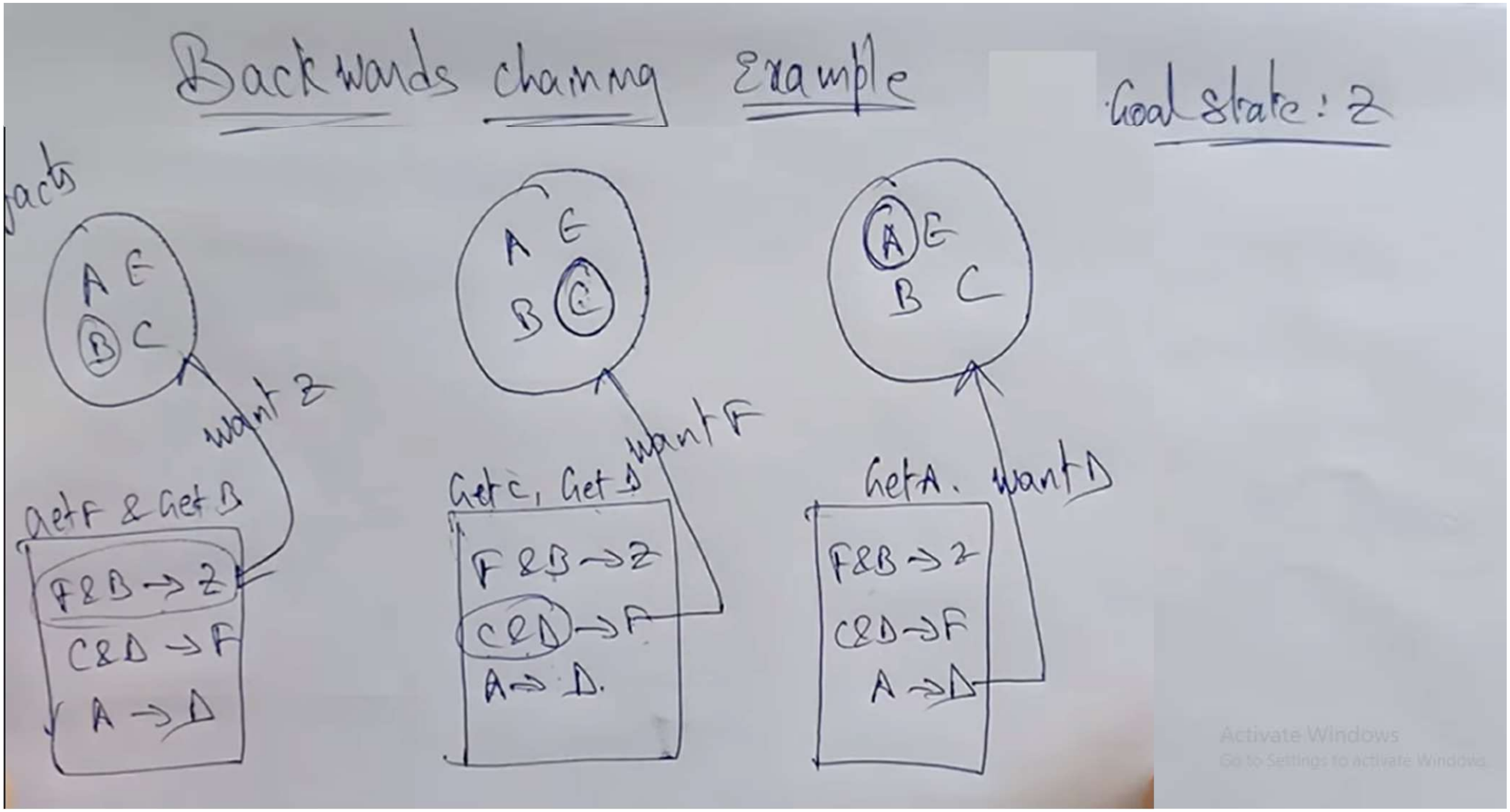
So next step: Proceed with “want F”

Backward Chaining



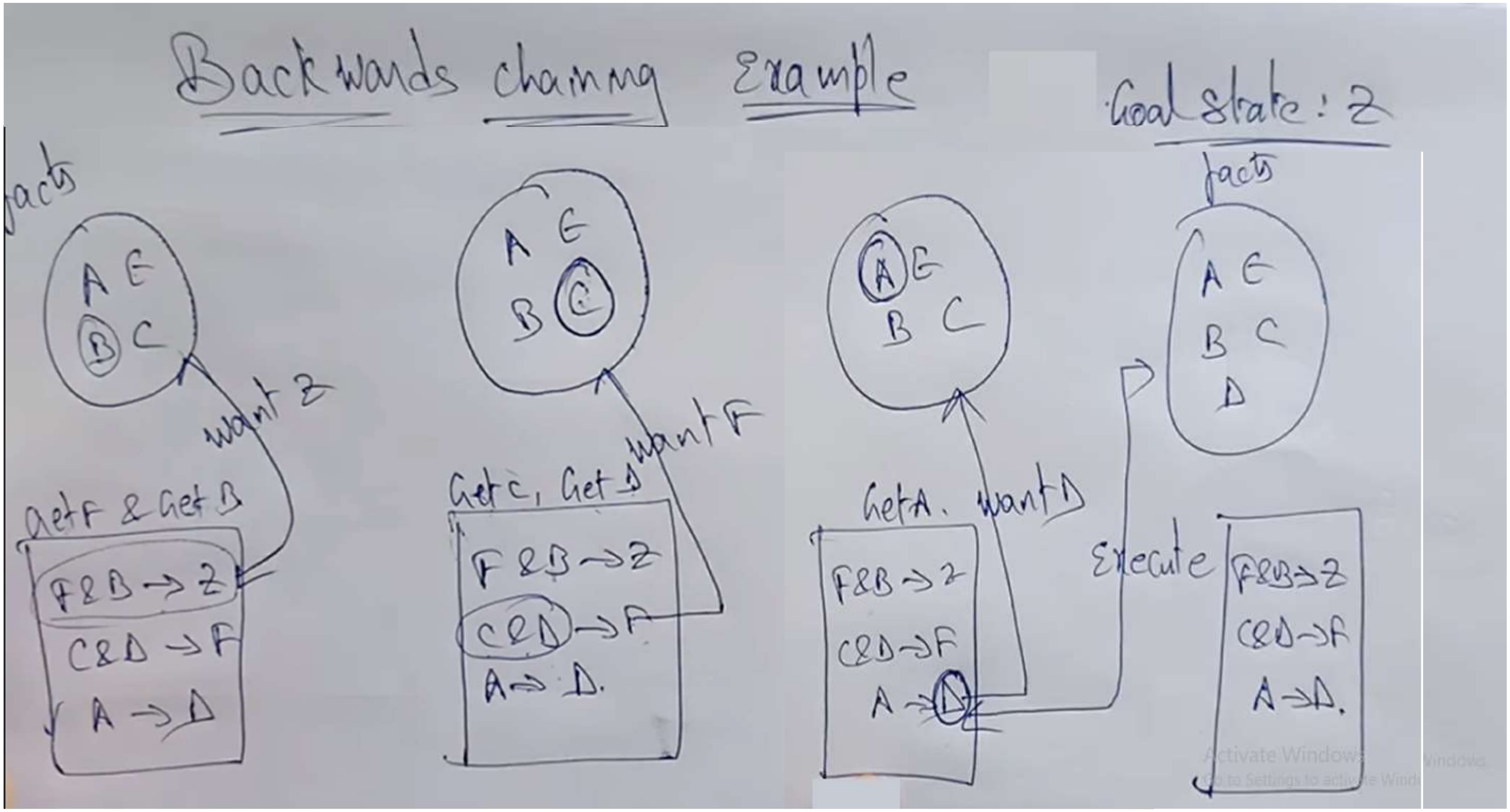
"Want F" - Whatever elements that are present in F (KB) is C & D
C is already present in DB (Facts)
So next step proceeds with "**Want D**"

Backward Chaining



"Want D" : where is D?
D is present in KB along with A: Get A (A is already in the Facts)
Now, One condition is over. Next start the Execution

Backward Chaining

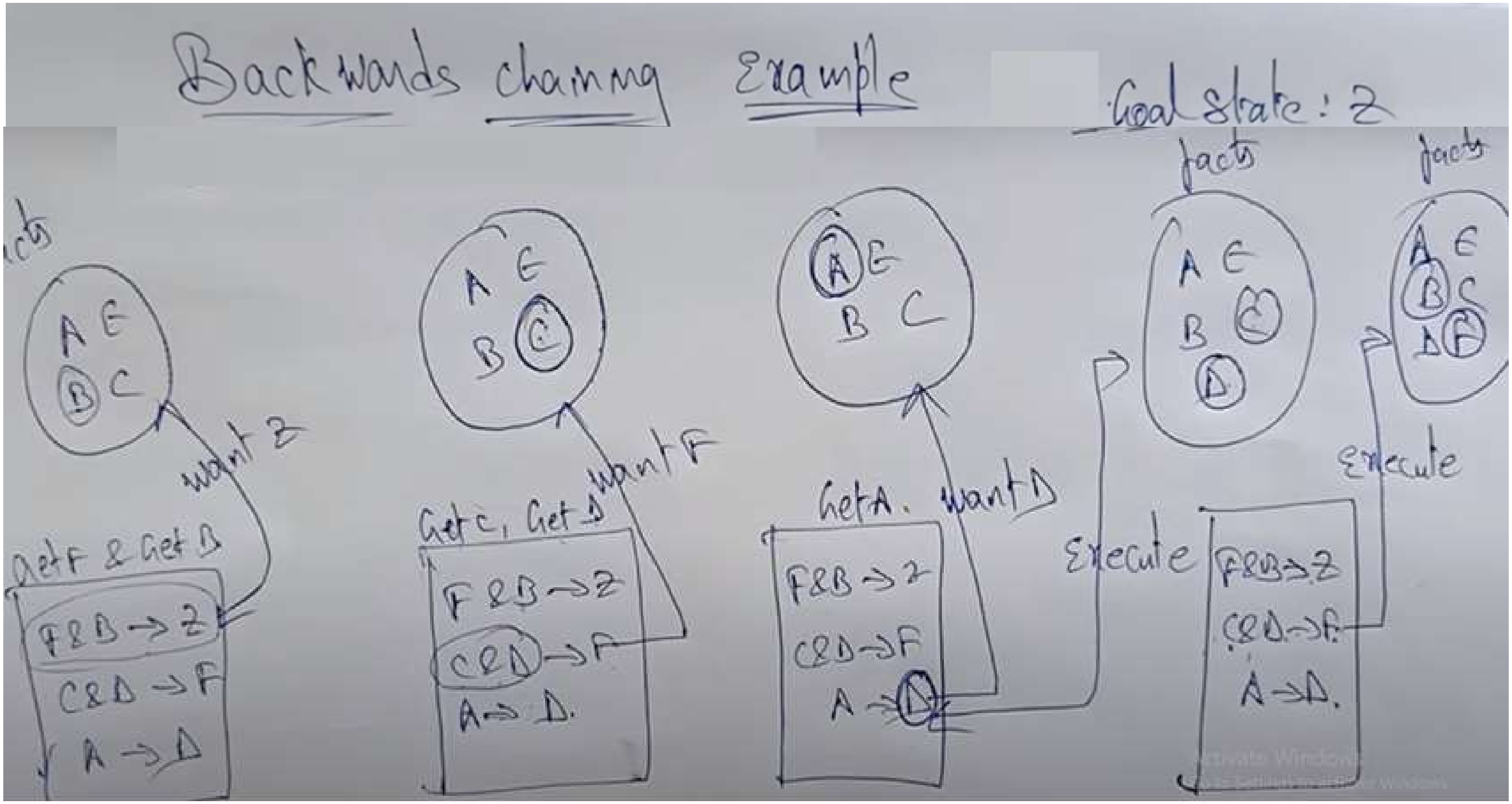


Next start the Execution

Take A-> D and start execute. i.e. Need to place this in Facts(DB)

What element to be placed? **D**

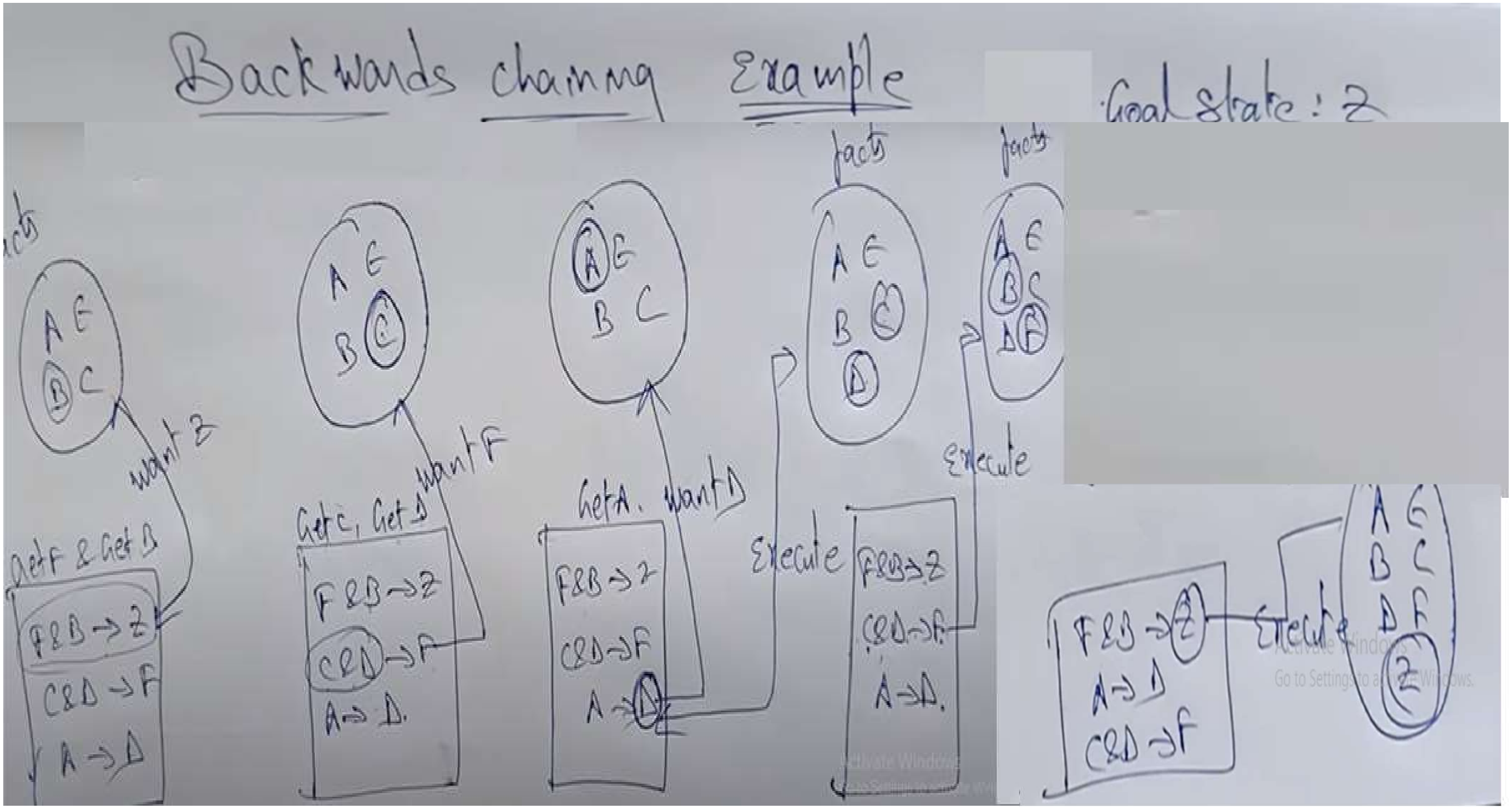
Backward Chaining



Next start the Execution at F

Check which rule is matching? F & B is matching

Backward Chaining



Next start the Execution at F

Check which rule is matching? F & B is matching

F & B executes Z . Now the Facts (DB) contains A, B, C, D, F & Z which is the GOAL

Backward Chaining

- These algorithms work backward from the goal, chaining through rules to find known facts that support the proof.
- The algorithm FOL-BC-ASK(KB, goal) will be proved if the knowledge base contains a clause of the form lhs ⇒ goal, where lhs (left-hand side) is a list of conjuncts.
- An atomic fact like American(West) is considered as a clause whose lhs is the empty list.

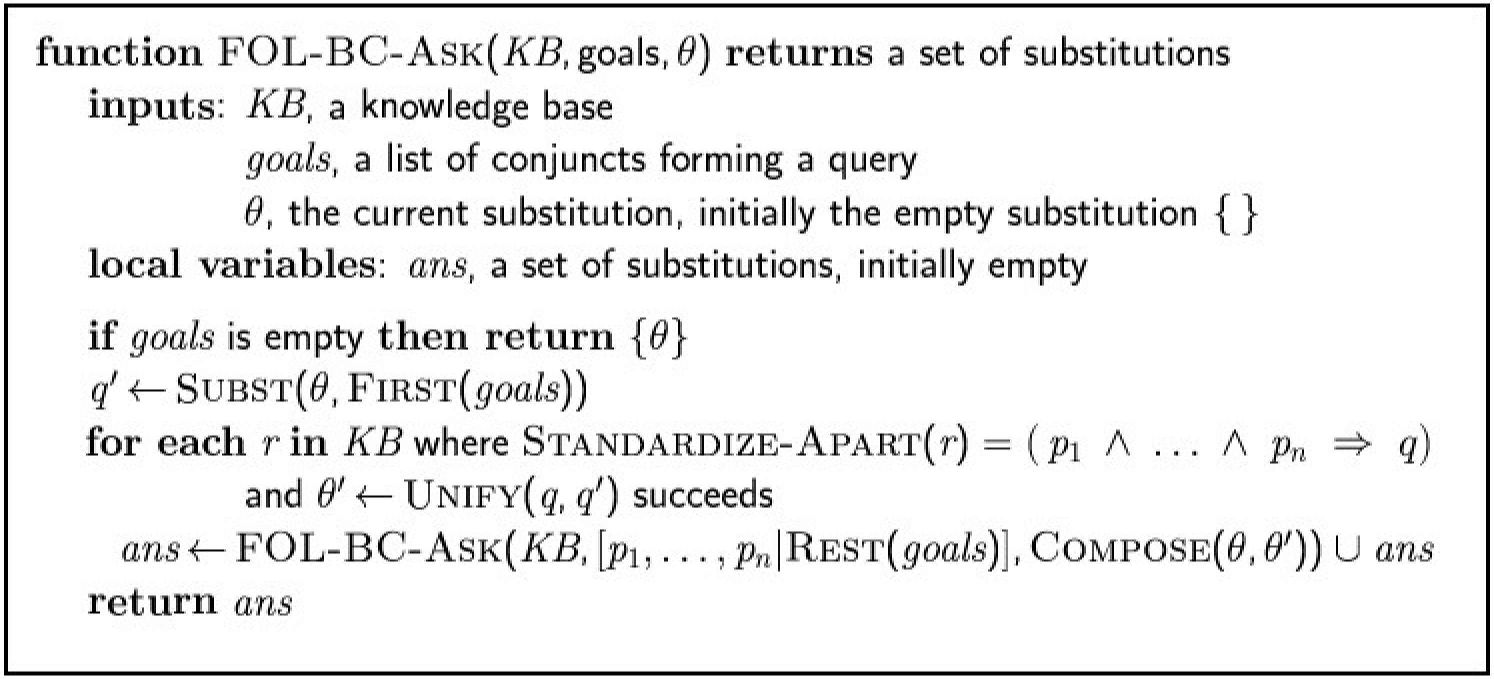
Backward Chaining

- For example, the query Person(x) could be proved with the substitution {x/John} as well as with {x/Richard }.
- The algorithm is implemented as a **generator** which is a function that returns multiple times, each time giving one possible result
- It is a kind of AND/OR search
  - the OR part because the goal query can be proved by any rule in the knowledge base,
  - AND part because all the conjuncts in the lhs of a clause must be proved.

Backward Chaining

- Backward chaining is clearly a depth-first search algorithm.
- So its space requirements are linear in the size of the proof.
- It also means that backward chaining suffers from problems with repeated states and incompleteness.

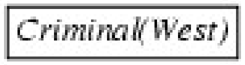
Backward chaining algorithm



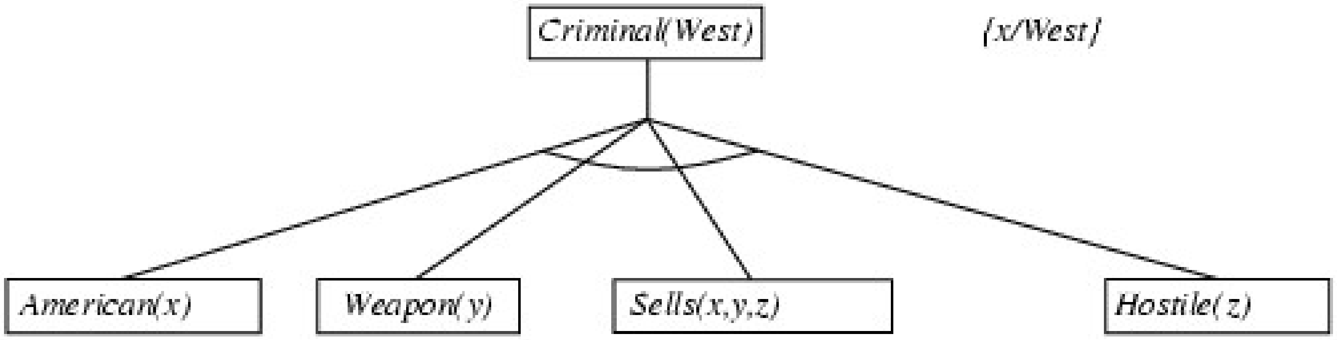
```
function FOL-BC-ASK($KB$, goals, $\theta$) returns a set of substitutions
   inputs: $KB$, a knowledge base
           $goals$, a list of conjuncts forming a query
           $\theta$, the current substitution, initially the empty substitution $\{\,\}$
   local variables: $ans$, a set of substitutions, initially empty

   if $goals$ is empty then return $\{\theta\}$
   $q' \leftarrow$ SUBST($\theta$, FIRST($goals$))
   for each $r$ in $KB$ where STANDARDIZE-APART($r$) $= (\,p_1 \wedge \ldots \wedge p_n \Rightarrow q)$
           and $\theta' \leftarrow$ UNIFY($q, q'$) succeeds
     $ans \leftarrow$ FOL-BC-ASK($KB$, $[p_1, \ldots, p_n|$REST($goals$)$]$, COMPOSE($\theta, \theta'$)) $\cup\ ans$
   return $ans$
```

SUBST(COMPOSE(θ_1, θ_2), p) = SUBST(θ_2, SUBST(θ_1, p))

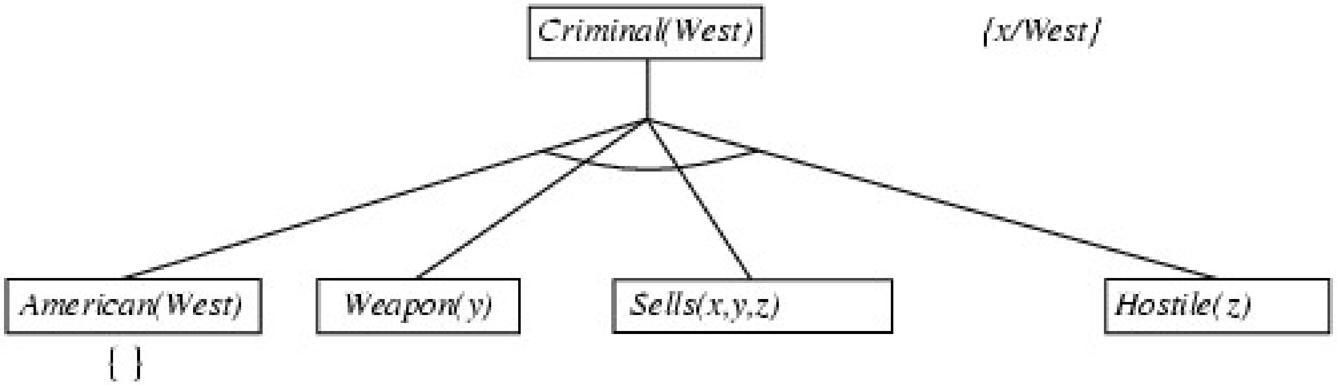
Backward chaining example



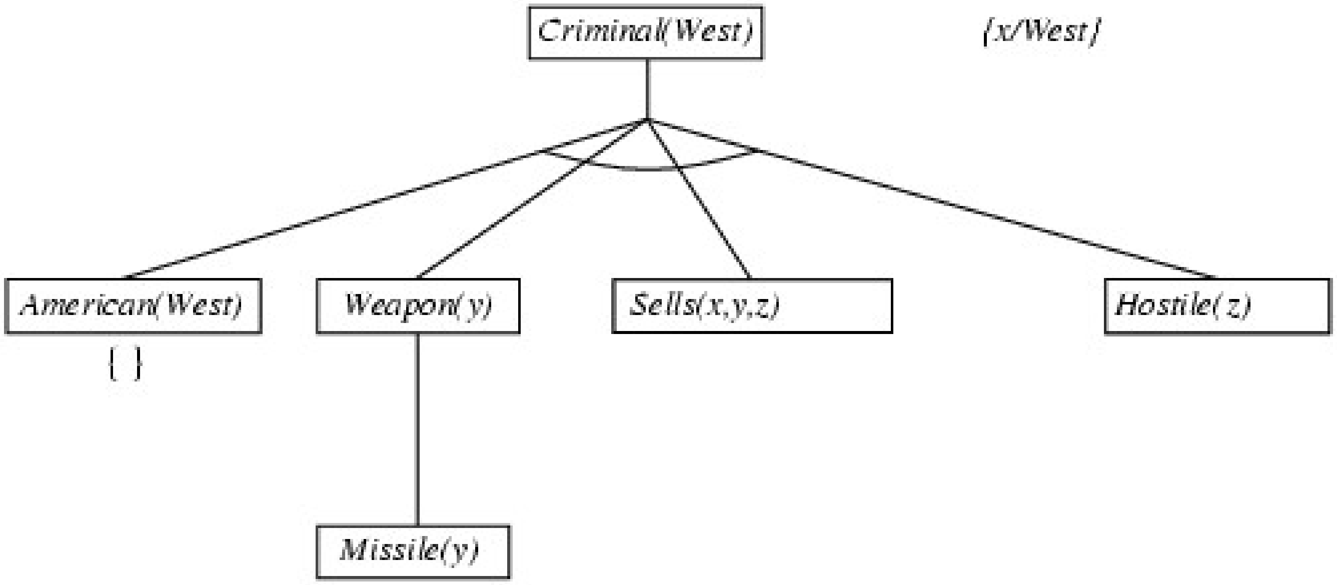
Backward chaining example



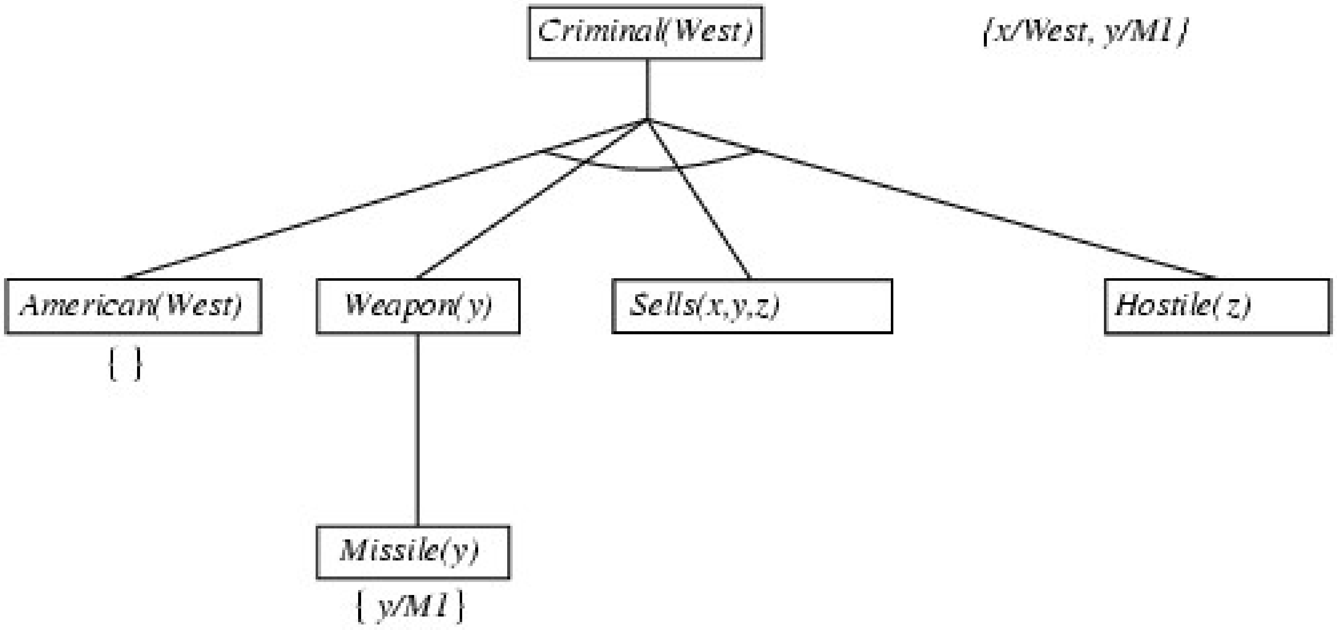
Backward chaining example



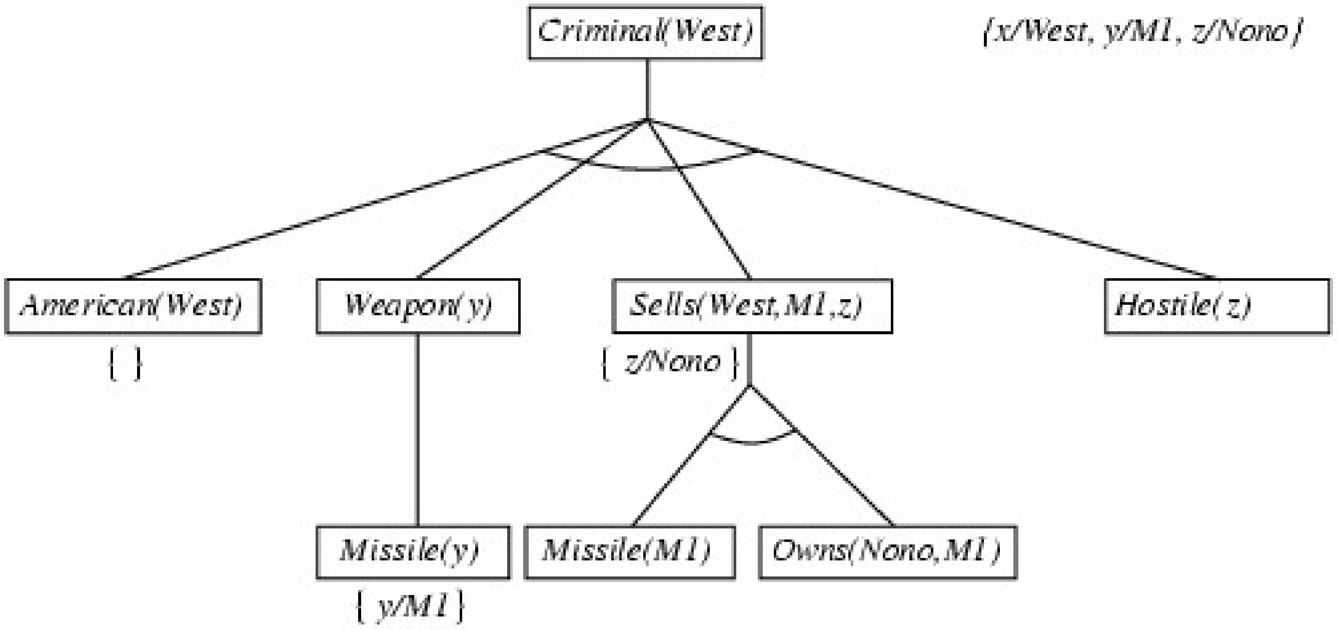
Backward chaining example



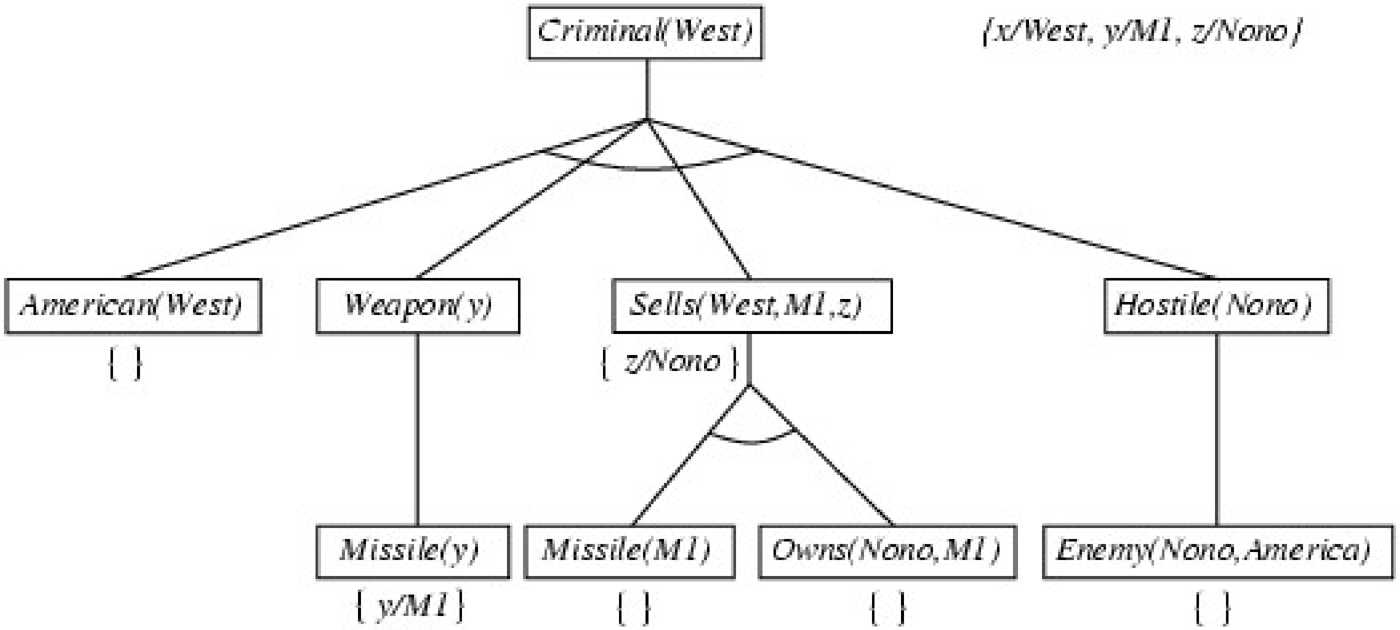
Backward chaining example



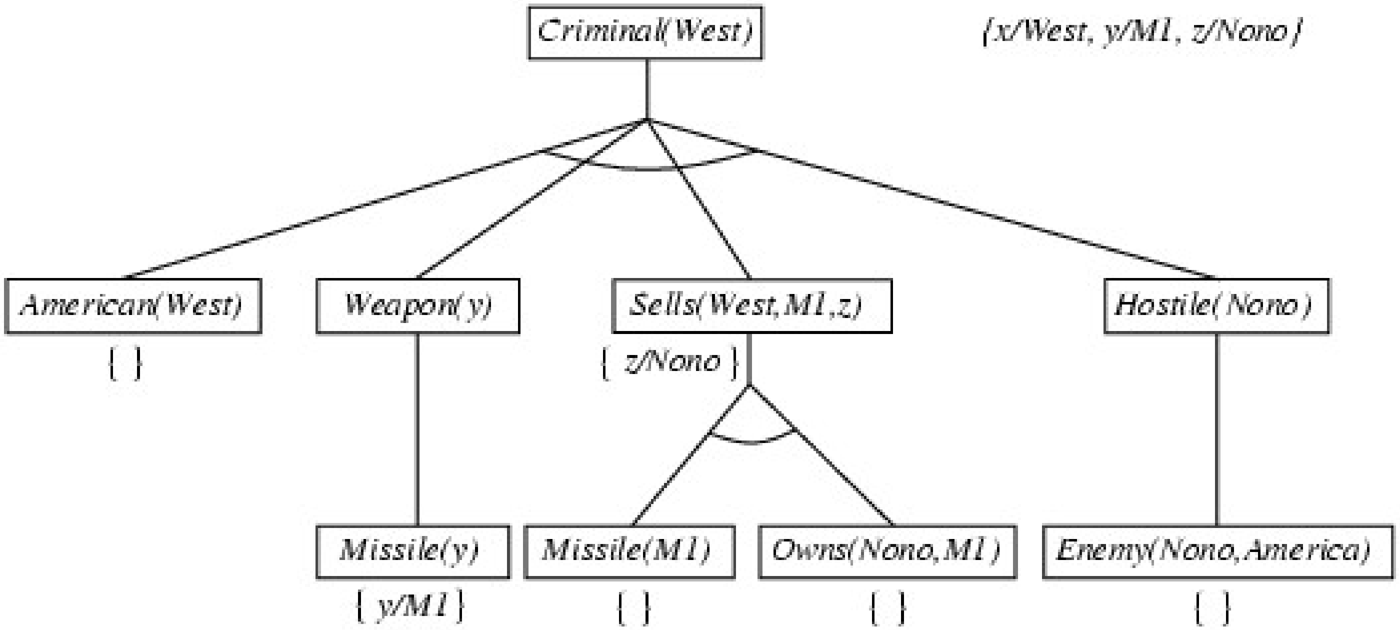
Backward chaining example



Backward chaining example



Backward chaining example



Properties of backward chaining

- Depth-first recursive proof search: space is linear in size of proof
-
- Incomplete due to infinite loops
-
  - ⇒ fix by checking current goal against every goal on stack
  -
- Inefficient due to repeated subgoals (both success and failure)
  - ⇒ fix using caching of previous results (extra space)
  -
- Widely used for logic programming