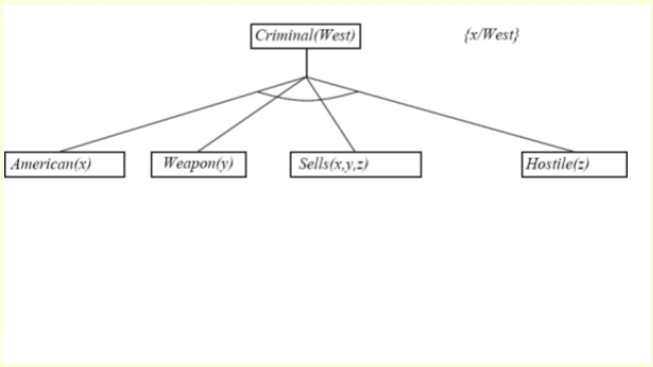
Backward Chaining

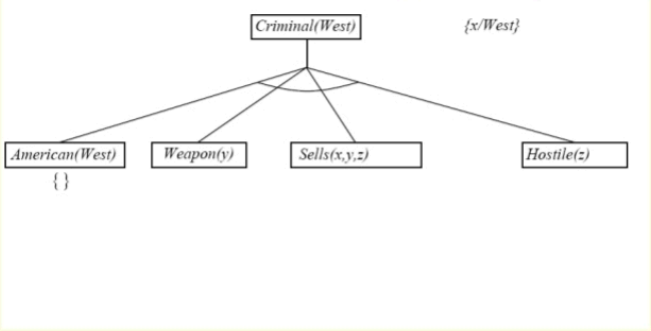
- Consider the item to be proven a goal
- Find a rule whose head is the goal (and bindings)
- Apply bindings to the body, and prove these (subgoals) in turn
- If you prove all the subgoals, increasing the binding set as you go, you will prove the item.
- Logic Programming (cprolog, on CS)

Backward Chaining Example

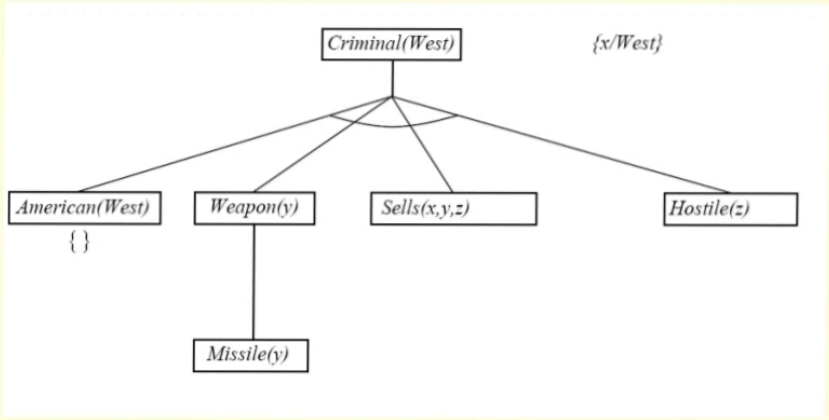
Backward Chaining Example



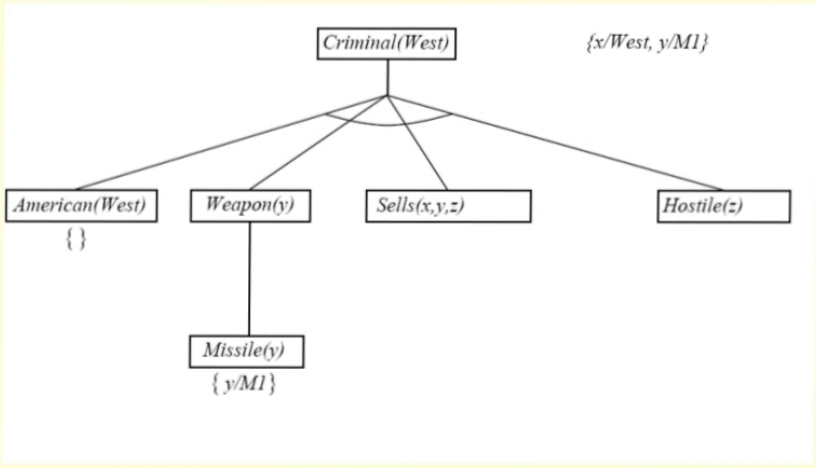
Backward Chaining Example



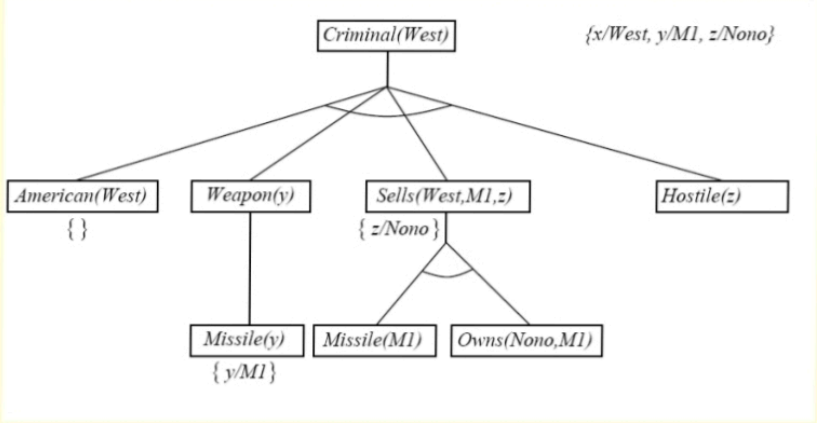
Backward Chaining Example



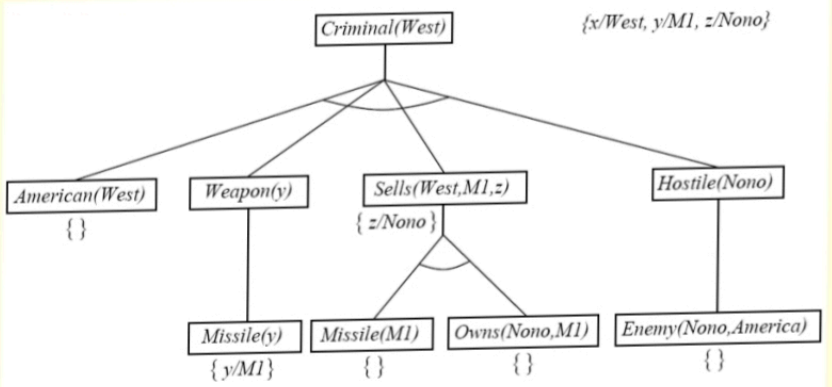
Backward Chaining Example



Backward Chaining Example



Backward Chaining Example



Backward Chaining Algorithm

function FOL-BC-ASK(KB, goals, θ) **returns** a set of substitutions
  inputs: KB, a knowledge base
    $goals$, a list of conjuncts forming a query
    θ, 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(θ, 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($KB, [p_1, \ldots, p_n | \text{REST}(goals)], \text{COMPOSE}(\theta', \theta)) \cup ans$
  return ans

operties of Backward Chainin

- Depth-first recursive proof search: space is linear in size of proof
- Incomplete due to infinite loops
  - Fix by checking current goal with every subgoal on the stack
- Inefficient due to repeated subgoals (both success and failure)
  - Fix using caching of previous results (extra space)
- Widely used without improvements for logic programming