The second major family of logical inference algorithms uses the **backward chaining** approach introduced in Section 7.5. These algorithms work backward from the goal, chaining through rules to find known facts that support the proof. We describe the basic algorithm, and then we describe how it is used in **logic programming**, which is the most widely used form of automated reasoning. We will also see that backward chaining has some disadvantages compared with forward chaining, and we look at ways to overcome them. Finally, we will look at the close connection between logic programming and constraint satisfaction problems.

A backward chaining algorithm

Figure 9.6 shows a simple backward-chaining algorithm, FOL-BC-ASK. It is called with a list of goals containing a single element, the original query, and returns the set of all substitutions satisfying the query. The list of goals can be thought of as a "stack" waiting to be worked on; if *all* of them can be satisfied, then the current branch of the proof succeeds. The algorithm takes the first goal in the list and finds every clause in the knowledge base whose positive literal, or **head**, unifies with the goal. Each such clause creates a new recursive call in which the premise, or **body**, of the clause is added to the goal stack. Remember that facts are clauses with a head but no body, so when a goal unifies with a known fact, no new subgoals are added to the stack and the goal is solved. Figure 9.7 is the proof tree for deriving $Criminal(West)$ from sentences (9.3) through (9.10).

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 already applied)
    θ, the current substitution, initially the empty substitution $\{\}$
  local variables: $answers$, a set of substitutions, initially empty

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

Figure 9.6 A simple backward-chaining algorithm.

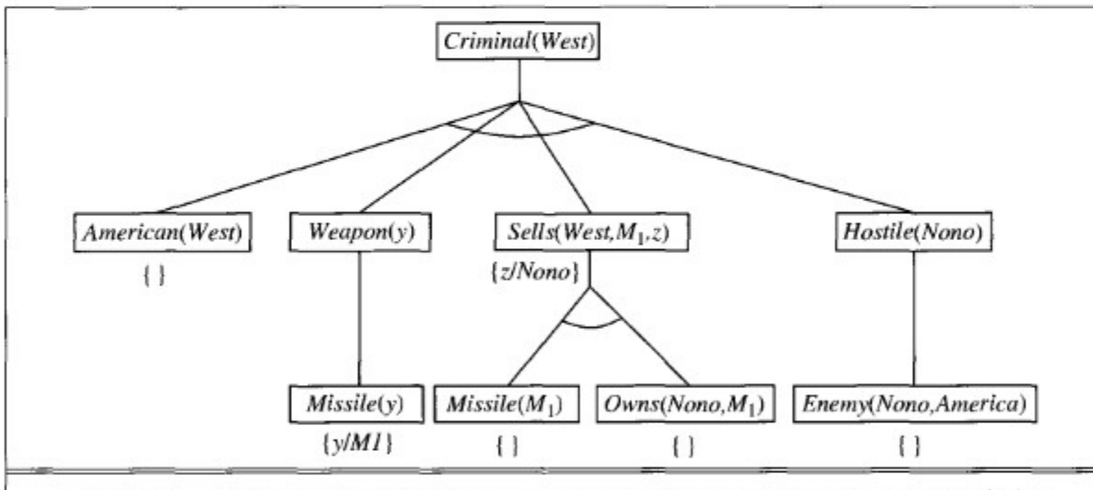

Figure 9.7 Proof tree constructed by backward chaining to prove that West is a criminal. The tree should be read depth first, left to right. To prove $Criminal(West)$, we have to prove the four conjuncts below it. Some of these are in the knowledge base, and others require further backward chaining. Bindings for each successful unification are shown next to the corresponding subgoal. Note that once one subgoal in a conjunction succeeds, its substitution is applied to subsequent subgoals. Thus, by the time FOL-BC-ASK gets to the last conjunct, originally $Hostile(z)$, z is already bound to $Nono$.

The algorithm uses **composition** of substitutions. COMPOSE(θ_1, θ_2) is the substitution whose effect is identical to the effect of applying each substitution in turn. That is,

$$\text{SUBST}(\text{COMPOSE}(\theta_1, \theta_2), p) = \text{SUBST}(\theta_2, \text{SUBST}(\theta_1, p)) \ .$$

In the algorithm, the current variable bindings, which are stored in θ, are composed with the bindings resulting from unifying the goal with the clause head, giving a new set of current bindings for the recursive call.