Backward-chaining algorithm

Backward chaining is designed to find all answers to a question posed to the knowledge base. Backward chaining therefore exhibits the functionality required for the ASK procedure. The backward-chaining algorithm BACK-CHAIN works by first checking to see if answers can be provided directly from sentences in the knowledge base. It then finds all implications whose conclusion unifies with the query, and tries to establish the premises of those implications, also by backward chaining. If the premise is a conjunction, then BACK-CHAIN processes the conjunction conjunct by conjunct, building up the unifier for the whole premise as it goes. The algorithm is shown in Figure 9.2. (Here UNIFY is assumed to standardize apart its second argument.)

Figure 9.3 is the proof tree for deriving *Criminal*(*West*) from sentences (9.22) through (9.30). As a diagram of the backward chaining algorithm, the tree should be read depth-first, left to right. To prove *Criminal*(x) we have to prove the five conjuncts below it. Some of these are in the knowledge base, and others require further backward chaining. Each leaf node has the substitution used to obtain it written below. Note that once one branch of a conjunction succeeds, its substitution is applied to subsequent branches. Thus, by the time BACK-CHAIN gets to *Sells*(x, z, y), all the variables are instantiated. Figure 9.3 can also be seen as a diagram of forward chaining. In this interpretation, the premises are added at the bottom, and conclusions are added once all their premises are in the *KB*. Figure 9.4 shows what can happen if an incorrect choice is made in the search—in this case, choosing America as the nation in question. There is no way to prove that America is a hostile nation, so the proof fails to go through, and we have to back up and consider another branch in the search space.

```
function BACK-CHAIN(KB, q) returns a set of substitutions

  BACK-CHAIN-LIST(KB, [q], {})
-----------------------------------------------------------------
function BACK-CHAIN-LIST(KB, qlist, θ) returns a set of substitutions
  inputs: KB, a knowledge base
          qlist, a list of conjuncts forming a query (θ already applied)
          θ, the current substitution
  local variables: answers, a set of substitutions, initially empty

  if qlist is empty then return {θ}
  q ← FIRST(qlist)
     for each q'_i in KB such that θ_i ← UNIFY(q, q'_i) succeeds do
        Add COMPOSE(θ, θ_i) to answers
     end
     for each sentence (p_1 ∧ ... ∧ p_n ⇒ q'_i) in KB such that θ_i ← UNIFY(q, q'_i) succeeds do
        answers ← BACK-CHAIN-LIST(KB, SUBST(θ_i, [p_1 ... p_n]), COMPOSE(θ, θ_i)) ∪ answers
     end
  return the union of BACK-CHAIN-LIST(KB, REST(qlist), θ) for each θ ∈ answers
```

Figure 9.2 The backward-chaining algorithm.