function FOL-FC-ASK(KB, α) returns a substitution or false
inputs: KB, the knowledge base, a set of first-order definite clauses
α, the query, an atomic sentence
local variables: new, the new sentences inferred on each iteration
repeat until new is empty
    new ← {} for each rule in KB do
    (p₁ ∧ ... ∧ pₙ ⇒ q) ← STANDARDIZE-VARIABLES(rule)
    for each θ such that SUBST(θ, p₁ ∧ ... ∧ pₙ) = SUBST(θ, p′₁ ∧ ... ∧ p′ₙ) for some p′₁, ..., p′ₙ in KB
        q′ ← SUBST(θ, q)
        if q′ does not unify with some sentence already in KB or new then
            add q′ to new
            φ ← UNIFY(q′, α)
            if φ is not fail then return φ
    add new to KB
return false

Figure 9.3 A conceptually straightforward, but very inefficient, forward-chaining algorithm. On each iteration, it adds to KB all the atomic sentences that can be inferred in one step from the implication sentences and the atomic sentences already in KB. The function STANDARDIZE-VARIABLES replaces all variables in its arguments with new ones that have not been used before.

function FOL-BC-ASK(KB, query) returns a generator of substitutions
return FOL-BC-OR(KB, query, { })

generator FOL-BC-OR(KB, goal, θ) yields a substitution
for each rule (lhs ⇒ rhs) in FETCH-RULES-FOR-GOAL(KB, goal) do
    (lhs, rhs) ← STANDARDIZE-VARIABLES((lhs, rhs))
    for each θ′ in FOL-BC-AND(KB, lhs, UNIFY(rhs, goal, θ)) do
        yield θ′

generator FOL-BC-AND(KB, goals, θ) yields a substitution
if θ = failure then return
else if LENGTH(goals) = 0 then yield θ
else do
    first, rest ← FIRST(goals), REST(goals)
    for each θ′ in FOL-BC-OR(KB, SUBST(θ, first), θ) do
        for each θ″ in FOL-BC-AND(KB, rest, θ′) do
            yield θ″

Figure 9.6 A simple backward-chaining algorithm for first-order knowledge bases.