function FOL-FC-ASK(KB, α) \textbf{returns} a substitution or \textit{false}  
inputs: KB, the knowledge base, a set of first-order definite clauses  
        α, the query, an atomic sentence  
local variables: \textit{new}, the new sentences inferred on each iteration  
repeat until \textit{new} is empty  
    \textit{new} ← {}  
    for each \textit{rule} in KB do  
        \textit{p}_1 \wedge \ldots \wedge \textit{p}_n \Rightarrow \textit{q} ← \text{STANDARDIZE-VARIABLES}(\textit{rule})  
        for each \textit{θ} such that \text{SUBST}(\textit{θ}, \textit{p}_1 \wedge \ldots \wedge \textit{p}_n) = \text{SUBST}(\textit{θ}, \textit{p}_1' \wedge \ldots \wedge \textit{p}_n')  
            for some \textit{p}_1', \ldots, \textit{p}_n' in \textit{KB}  
            \textit{q}' ← \text{SUBST}(\textit{θ}, \textit{q})  
            if \textit{q}' does not unify with some sentence already in \textit{KB} or \textit{new} then  
                add \textit{q}' to \textit{new}  
                \textit{φ} ← \text{UNIFY}(\textit{q}', α)  
                if \textit{φ} is not \textit{fail} then return \textit{φ}  
        add \textit{new} to \textit{KB}  
return \textit{false}  

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

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

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

generator FOL-BC-AND(KB, goals, θ) \textbf{yields} a substitution  
if θ = \text{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, \text{SUBST}(θ, first), θ) do  
        for each θ'' in FOL-BC-AND(KB, rest, θ') do  
            yield θ''  

\textbf{Figure 9.6}  A simple backward-chaining algorithm for first-order knowledge bases.