Let α be a set of attributes. We call the set of all attributes functionally determined by α under a set F of functional dependencies the closure of α under F; we denote it by α^+. Figure 7.8 shows an algorithm, written in pseudocode, to compute α^+. The input is a set F of functional dependencies and the set α of attributes. The output is stored in the variable *result*.

To illustrate how the algorithm works, we shall use it to compute $(AG)^+$ with the functional dependencies defined in Section 7.4.1. We start with *result* $= AG$. The first time that we execute the **repeat** loop to test each functional dependency, we find that:

- $A \rightarrow B$ causes us to include B in *result*. To see this fact, we observe that $A \rightarrow B$ is in F, $A \subseteq$ *result* (which is AG), so *result* := *result* $\cup B$.
- $A \rightarrow C$ causes *result* to become $ABCG$.
- $CG \rightarrow H$ causes *result* to become $ABCGH$.
- $CG \rightarrow I$ causes *result* to become $ABCGHI$.

The second time that we execute the **repeat** loop, no new attributes are added to *result*, and the algorithm terminates.

Let us see why the algorithm of Figure 7.8 is correct. The first step is correct because $\alpha \rightarrow \alpha$ always holds (by the reflexivity rule). We claim that, for any subset β of *result*, $\alpha \rightarrow \beta$. Since we start the **repeat** loop with $\alpha \rightarrow$ *result* being true, we can add γ to *result* only if $\beta \subseteq$ *result* and $\beta \rightarrow \gamma$. But then *result* $\rightarrow \beta$ by the reflexivity rule, so $\alpha \rightarrow \beta$ by transitivity. Another application of transitivity shows that $\alpha \rightarrow \gamma$ (using $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$). The union rule implies that $\alpha \rightarrow$ *result* $\cup\, \gamma$, so α functionally determines any new result generated in the **repeat** loop. Thus, any attribute returned by the algorithm is in α^+.

It is easy to see that the algorithm finds all of α^+. Consider an attribute A in α^+ that is not yet in *result* at any point during the execution. There must be a way to prove that *result* $\rightarrow A$ using the axioms. Either *result* $\rightarrow A$ is in F itself (making the proof trivial and ensuring A is added to *result*) or there must a proof step using transitivity to show

```
result := α;
repeat
    for each functional dependency β → γ in F do
      begin
         if β ⊆ result then result := result ∪ γ;
      end
until (result does not change)
```

Figure 7.8 An algorithm to compute α^+, the closure of α under F.