$F_c = F$
repeat
    Use the union rule to replace any dependencies in F_c of the form
        $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1 \beta_2$.
    Find a functional dependency $\alpha \rightarrow \beta$ in F_c with an extraneous
        attribute either in α or in β.
        /* Note: the test for extraneous attributes is done using F_c, not F */
    If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$ in F_c.
until (F_c does not change)

Figure 7.9 Computing canonical cover.

Having defined the concept of extraneous attributes, we can explain how we can construct a simplified set of functional dependencies equivalent to a given set of functional dependencies.

A **canonical cover** F_c for F is a set of dependencies such that F logically implies all dependencies in F_c, and F_c logically implies all dependencies in F. Furthermore, F_c must have the following properties:

- No functional dependency in F_c contains an extraneous attribute.
- Each left side of a functional dependency in F_c is unique. That is, there are no two dependencies $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ in F_c such that $\alpha_1 = \alpha_2$.

A canonical cover for a set of functional dependencies F can be computed as described in Figure 7.9. It is important to note that when checking if an attribute is extraneous, the check uses the dependencies in the current value of F_c, and **not** the dependencies in F. If a functional dependency contains only one attribute in its right-hand side, for example $A \rightarrow C$, and that attribute is found to be extraneous, we would get a functional dependency with an empty right-hand side. Such functional dependencies should be deleted.

Since the algorithm permits a choice of any extraneous attribute, it is possible that there may be several possible canonical covers for a given F. Any such F_c is equally acceptable. Any canonical cover of F, F_c, can be shown to have the same closure as F; hence, testing whether F_c is satisfied is equivalent to testing whether F is satisfied. However, F_c is minimal in a certain sense—it does not contain extraneous attributes, and it combines functional dependencies with the same left side. It is cheaper to test F_c than it is to test F itself.

We now consider an example. Assume we are given the following set F of functional dependencies on schema (A, B, C):