```
First sets for all symbols:
    for all terminals a, FIRST(a) := {a}
    for all nonterminals X, FIRST(X) := ∅
    for all productions X ⟶ ϵ, add ϵ to FIRST(X)
    repeat
        (outer) for all productions X ⟶ Y1 Y2 ... Yk,
            (inner) for i in 1..k
                add (FIRST(Yi) \ {ϵ}) to FIRST(X)
                if ϵ ∉ FIRST(Yi) (yet)
                    continue outer loop
            add ϵ to FIRST(X)
    until no further progress

First set subroutine for string X1 X2 ... Xn, similar to inner loop above:
    return_value := ∅
    for i in 1..n
        add (FIRST(Xi) \ {ϵ}) to return_value
        if ϵ ∉ FIRST(Xi)
            return
    add ϵ to return_value

Follow sets for all symbols:
    FOLLOW(S) := {ϵ}, where S is the start symbol
    for all other symbols X, FOLLOW(X) := ∅
    repeat
        for all productions A ⟶ α B β,
            add (FIRST(β) \ {ϵ}) to FOLLOW(B)
        for all productions A ⟶ α B
                or A ⟶ α B β, where ϵ ∈ FIRST(β),
            add FOLLOW(A) to FOLLOW(B)
    until no further progress

Predict sets for all productions:
    for all productions A ⟶ α
        PREDICT(A ⟶ α) := (FIRST(α) \ {ϵ})
            ∪ (if ϵ ∈ FIRST(α) then FOLLOW(A) else ∅)
```

Figure 2.23 Algorithm to calculate FIRST, FOLLOW, and PREDICT sets. The grammar is LL(1) if and only if the PREDICT sets are disjoint.

(i.e., converge on a solution), because the sizes of the sets are bounded by the number of terminals in the grammar.

If in the process of calculating PREDICT sets we find that some token belongs to the PREDICT set of more than one production with the same left-hand side, then the grammar is not LL(1), because we will not be able to choose which of the productions to employ when the left-hand side is at the top of the parse stack (or we are in the left-hand side's subroutine in a recursive descent parser) and we see the token coming up in the input. This sort of ambiguity is known as a *predict-predict conflict*; it can arise either because the same token can begin