Follow

During the derivation of a string in a grammar, there are many intermediate strings, which contain non-terminals as well as terminals. For example, the grammar

T → aTc | R
R → bR | λ

generates the language $\{a^i b^j c^i \mid i,j \geq 0\}$. A derivation for aabcc looks like: T → aTc → aaTcc → aaRcc → aabRcc → aabcc

The Follow set of a non-terminal is the set of terminals that *can* appear adjacent-to-the-right (ie, follows) it in an intermediate string at any time during a derivation. We can see from the above derivation that c can follow T and that c can follow R. That means c ∈ Follow(T) and c ∈ Follow(R).

Rather than generate a large number of derivations and inspect each to figure out the Follow sets, we can deduce them from the productions of the grammar. The strategy is to find set statements (like those seen in my explanation of First sets), and then make minimal sets that satisfy the statements.

To find the Follow set statements, first add a new production S' → S$, where S is whatever is the start symbol of your grammar and $ is a new terminal not seen elsewhere in your grammar. This $ symbol represents the end of your input token sequence. We do this because we want to be able to represent that the end-of-input can follow a non-terminal. Next use these rules to generate the set statements:

1. A production right-hand-side containing Xy anywhere in it, where X is a non-terminal and y is a terminal creates the statement y ∈ Follow(X).
2. A production right-hand-side containing XY anywhere in it, where X and Y are both non-terminals creates a set statement First(Y) ⊆ Follow(X). This is because any symbol that can start a string in L(Y) can follow an X during a derivation.
3. If a production for X has a right-hand-side ending in a non-terminal Y, then Follow(X) ⊆ Follow(Y). To see why, imagine an intermediate derivation string has X followed by terminal c (ie, ...Xc...). If the production is applied to this X, then the X gets turned into something ending in Y (ie, ...Yc...). Whatever symbol had been following X is now following Y. Thus, Follow(X) ⊆ Follow(Y).

Note carefully: In all three of these rules you must consider what would happen if a nullable non-terminal is substituted with an empty string. If a non-terminal is nullable you must apply all of the above rules again with the nullable non-terminal deleted. For example, if XYZ is on the right-hand-side of a production you get statements First(Y) ⊆ Follow(X) and First(Z) ⊆ Follow(Y), but if Y is nullable you also get First(Z) ⊆ Follow(X) because the Y could become the empty string and then you have Z following X.

Example 1:

Let's find the Follow set statements for A → BCdEF given that B, C, E, F are all non-nullable.

Rule 1 gives us d ∈ Follow(C)

Rule 2 gives us First(C) ⊆ Follow(B) and First(F) ⊆ Follow(E)

Rule 3 gives us Follow(A) ⊆ Follow(F)

Example 2:

Let's find the Follow set statements for A → BCdEF given that B, C, E, F are all nullable.

In addition to the four statements from Example 1, we consider new statements that arise when we allow any combination of B, C, E, F to become empty strings.

Rule 1 gives us d ∈ Follow(B) (when C goes to the empty string)

Rule 2 gives us nothing new because deleting any of the non-terminals does not create any new pairs of adjacent non-terminals.

Rule 3 gives us Follow(A) ⊆ Follow(E) (when F goes to the empty string)

Once you have your set statements, you want to find the smallest sets that satisfy them. Here's a fixed-point algorithm that does so.

1. Make a list of all non-terminals and seed their Follow sets with the $\in$ statements.
2. For each $\subseteq$ statement, make it true by copying any missing elements from the set on the left-hand-side of the statement to the set on the right-hand-side of the statement.
3. If Step 2 changed anything, do Step 2 again.

Once Step 2 does not change anything, you have reached a "fixed point" in the algorithm and you should stop (because further iterations won't change anything either). Your final list of sets tells you the Follow set of each non-terminal.

As an example, consider the grammar above. With T' $\rightarrow$ T$ added during the statement-finding phase, the statements we find using the above rules are

Rule 1 (terminal following non-terminal): $ $\in$ Follow(T) and c $\in$ Follow(T)

Rule 2 (non-terminal following non-terminal): none

Rule 3 (non-terminal at end of right-hand-side): Follow(T) $\subseteq$ Follow(R) and Follow(R) $\subseteq$ Follow(R)

Now to find the smallest sets that satisfy these statements. After seeding Follow(T) = {c,$} and Follow(R) = { }. After one iteration of Step 2, Follow(T) = {c,$} and Follow(R) = {c,$}. After a third iteration Follow(T) = {c,$} and Follow(R) = {c,$} are still true, which means further iterations won't change anything and we're done.

Note that when a set statement is of the form A $\subseteq$ A, it is a tautology that does nothing for us. It is okay to ignore such statements.

$

What are the $ terminal and S' $\rightarrow$ S$ production all about? The sole purpose of this production is to force $ into your start symbol's Follow set, and the sole purpose of the $ token is to represent that the end-of-the token stream should follow the parsing of the start symbol. The $ symbol will not appear in any First set and can only appear in Follow sets. It's required because sometime the end of the token stream is a predictor for some production while parsing.

The $ is just conceptual however. Your scanner should provide some mechanism for letting you know there are no more tokens, and your parser should use this information to select the appropriate production to continue a correct parsing for the selector $.