**FOLLOW Set in Syntax Analysis**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 26 Jan, 2023

 Read

 Discuss

We have discussed the following topics on Syntax Analysis.

[Introduction to Syntax Analysis](https://www.geeksforgeeks.org/introduction-to-syntax-analysis-in-compiler-design/)   
[Why FIRST and FOLLOW?](https://www.geeksforgeeks.org/why-first-and-follow-in-compiler-design/)   
[FIRST Set in Syntax Analysis](https://www.geeksforgeeks.org/first-set-in-syntax-analysis/) 

FOLLOW set is a concept used in syntax analysis, specifically in the context of LR parsing algorithms. It is a set of terminals that can appear immediately after a given non-terminal in a grammar.

The FOLLOW set of a non-terminal A is defined as the set of terminals that can appear immediately after A in any derivation of the grammar. If A can appear at the right-hand side of a production rule, then the FOLLOW set of the left-hand side non-terminal of that production rule will be added to the FOLLOW set of A.

FOLLOW set is used in LR parsing to determine when to reduce a production rule. For example, if the next symbol in the input stream is in the FOLLOW set of a non-terminal, then that non-terminal can be safely reduced using the production rule that starts with that non-terminal.

To compute the FOLLOW set of a grammar, one can start with the FOLLOW set of the starting symbol being the EOF (End Of File) symbol and continue the process by adding the FOLLOW set of a non-terminal in the right-hand side of a production to the non-terminal in the left-hand side of the production. Repeat this process until no new element can be added to any set.

FOLLOW set is a fundamental concept in syntax analysis, and it is used in LR parsing algorithms. Its computation is a crucial step in the construction of LR parsing tables, which are used by LR parsers to parse input efficiently.

In this post, FOLLOW Set is discussed.

**Follow(X)** to be the set of terminals that can appear immediately to the right of Non-Terminal X in some sentential form.   
Example:

S ->Aa | Ac

A ->b

S S

/ \ / \

A a A c

| |

b b

Here, FOLLOW (A) = {a, c}

**Rules to compute FOLLOW set:**

1) FOLLOW(S) = { $ } // where S is the starting Non-Terminal

2) If A -> pBq is a production, where p, B and q are any grammar symbols,

then everything in FIRST(q) except Є is in FOLLOW(B).

3) If A->pB is a production, then everything in FOLLOW(A) is in FOLLOW(B).

4) If A->pBq is a production and FIRST(q) contains Є,

then FOLLOW(B) contains { FIRST(q) – Є } U FOLLOW(A)

**Example 1:**

**Production Rules:**

E -> TE’

E’ -> +T E’|Є

T -> F T’

T’ -> \*F T’ | Є

F -> (E) | id

**FIRST set**

FIRST(E) = FIRST(T) = { ( , id }

FIRST(E’) = { +, Є }

FIRST(T) = FIRST(F) = { ( , id }

FIRST(T’) = { \*, Є }

FIRST(F) = { ( , id }

**FOLLOW Set**

FOLLOW(E) = { $ , ) } // Note ')' is there because of 5th rule

FOLLOW(E’) = FOLLOW(E) = { $, ) } // See 1st production rule

FOLLOW(T) = { FIRST(E’) – Є } U FOLLOW(E’) U FOLLOW(E) = { + , $ , ) }

FOLLOW(T’) = FOLLOW(T) = { + , $ , ) }

FOLLOW(F) = { FIRST(T’) – Є } U FOLLOW(T’) U FOLLOW(T) = { \*, +, $, ) }

**Example 2:**

**Production Rules:**

S -> aBDh

B -> cC

C -> bC | Є

D -> EF

E -> g | Є

F -> f | Є

**FIRST set**

FIRST(S) = { a }

FIRST(B) = { c }

FIRST(C) = { b , Є }

FIRST(D) = FIRST(E) U FIRST(F) = { g, f, Є }

FIRST(E) = { g , Є }

FIRST(F) = { f , Є }

**FOLLOW Set**

FOLLOW(S) = { $ }

FOLLOW(B) = { FIRST(D) – Є } U FIRST(h) = { g , f , h }

FOLLOW(C) = FOLLOW(B) = { g , f , h }

FOLLOW(D) = FIRST(h) = { h }

FOLLOW(E) = { FIRST(F) – Є } U FOLLOW(D) = { f , h }

FOLLOW(F) = FOLLOW(D) = { h }

**Example 3:**

**Production Rules:**

S -> ACB|Cbb|Ba

A -> da|BC

B-> g|Є

C-> h| Є

**FIRST set**

FIRST(S) = FIRST(A) U FIRST(B) U FIRST(C) = { d, g, h, Є, b, a}

FIRST(A) = { d } U {FIRST(B)-Є} U FIRST(C) = { d, g, h, Є }

FIRST(B) = { g, Є }

FIRST(C) = { h, Є }

**FOLLOW Set**

FOLLOW(S) = { $ }

FOLLOW(A) = { h, g, $ }

FOLLOW(B) = { a, $, h, g }

FOLLOW(C) = { b, g, $, h }

**Note :**

1. Є as a FOLLOW doesn’t mean anything (Є is an empty string).
2. $ is called end-marker, which represents the end of the input string, hence used while parsing to indicate that the input string has been completely processed.
3. The grammar used above is Context-Free Grammar (CFG). The syntax of a programming language can be specified using CFG.
4. CFG is of the form A -> B, where A is a single Non-Terminal, and B can be a set of grammar symbols ( i.e. Terminals as well as Non-Terminals)