# Calculating FIRST and FOLLOW sets

S. Kamin

FIRST and FOLLOW sets are used when constructing recursive descent parsers (when the grammar is too complex to do it by inspection). They are also used in the construction of LR(1) parsers, although we are not covering that construction in class.

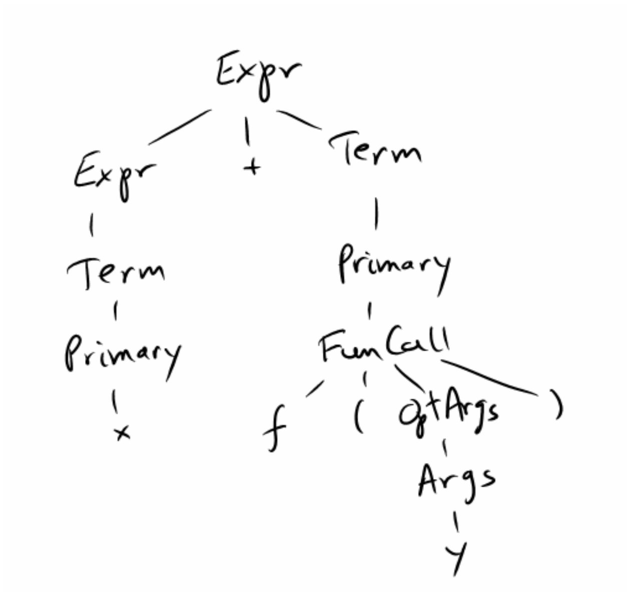
Simply put, for non-terminal A, FIRST(A) is the set of tokens that can occur as the first token in a string derivable from A, and FOLLOW(A) is the set of tokens that can occur immediately after A in a string derivable from the start symbol of the grammar. The purpose of this document is to formalize these definitions and illustrate how FIRST and FOLLOW sets are calculated.

Definition A *properly pruned* syntax tree is a syntax tree in which some (zero or more) nodes have had *all* their children removed. A *sentential form* is the frontier of a properly pruned syntax tree.

Example
We'll be using this grammar for examples in this document:

```
Expr -> Expr + Term | Term
Term -> Term * Primary | Primary
Primary -> FunCall | Id | ( Expr )
Funcall -> Id ( OptArgs )
OptArgs -> ε  |  Args
Args ->  Expr  |  Args , Expr
```

For example, given this syntax tree:



We can see that the following are sentential forms: x+f(y), Expr+Term, Expr+Funcall. On the other hand, Expr+f( can be obtained as the frontier of the tree if we remove *some* of the children of the FunCall node, but it is not a sentential form.

Definition *FOLLOW(A)* is the set of all tokens that can appear immediately after A in some sentential form. By convention, the set of tokens is assumed to include "eof", and each sentence is assumed to end with that token.

Definition *FIRST(A)* is the set of all tokens that can be the first token in a string derived from A (that is, in the frontier of a syntax tree rooted at A). For tokens t, we define *FIRST(A)* to be {t}. And for sequences of grammar symbols $X_1 ... X_n$ (for n >= 0), *FIRST($X_1 ... X_n$)* is the set of all tokens that can be the first token in a string derived from $X_1 ... X_n$; that is, the first token in a string obtained by concatenating the frontiers of syntax trees rooted at $X_1$, ..., $X_n$. In addition, if each of $X_1$ ... $X_n$ is a non-terminal that can produce the empty string, or if n = 0, we add the symbol • to the FIRST set.

In our grammar, we have:

FIRST(Expr) = FIRST(Term) = FIRST(Primary) = FIRST(Args) = { Id, ( }
FIRST(FunCall) = {Id}
FIRST(OptArgs) = { •, Id, ( }

FOLLOW(Expr) = { +, ), eof, ',' }
FOLLOW(Term) = FOLLOW(Primary) = FOLLOW(FunCall) = { +, *, ), eof }
FOLLOW(OptArgs) = { ) }
FOLLOW(Args) = { ), ',' }

To calculate FIRST and FOLLOW sets, we start with the following definitions:

Define FIRST: sequences of grammar symbols -> Tokens $\cup$ {•}:

FIRST($\varepsilon$) = {•}
FIRST(t) = {t} (for token t)
FIRST(A) = union of all sets FIRST($\alpha$) for all production A -> $\alpha$
FIRST($X_1 ... X_n$), for n>1, = FIRST($X_1$), if • $\notin$ FIRST($X_1$)
= FIRST($X_1$) - {•} $\cup$ FIRST($X_2 ... X_n$), o.w.

Note that in the final clause, we will have • in FIRST($X_1 ... X_n$) only if we have it in each set FIRST($X_1$), ..., FIRST($X_n$).

This definition cannot be circular if the grammar is LL(1) – since that would violate the prohibition against left recursion – but it can be circular in general. Since FIRST sets are useful even for non-LL(1) grammars, we need to explain how to use this definition to calculate FIRST sets when it is circular. An algorithm for calculating FIRST sets is:

1. Start by assuming FIRST(A) = $\varnothing$ for all A.

2. Using the values of FIRST(A) calculated thus far, use the definition above to calculate new values for FIRST(A).
3. Repeat this process until no new elements are added to any of the sets FIRST(A).

For example, we calculate the FIRST sets for the above grammar, in steps:

0. Start with FIRST(Expr) = FIRST(Term) = ... = ∅
1. FIRST(Expr) = FIRST(Expr + Term) ∪ FIRST(Term) = ∅
   FIRST(Term) = FIRST(Term + Primary) ∪ FIRST(Primary) = ∅
   FIRST(Primary) = FIRST(FunCall) ∪ FIRST(Id) ∪ FIRST( (Expr) )
   = ∅ ∪ {Id, (} = { Id, ( }
   FIRST(FunCall) = FIRST( Id ( OptArgs ) ) = {Id}
   FIRST(OptArgs) = FIRST(ε) ∪ FIRST(Args) = { • } ∪ ∅ = { • }
   FIRST(Args) = FIRST(Expr) ∪ FIRST(Args , Expr) = ∅ ∪ ∅ = ∅
2. FIRST(Expr) = FIRST(Expr + Term) ∪ FIRST(Term) = ∅
   FIRST(Term) = FIRST(Term + Primary) ∪ FIRST(Primary) = { Id, ( }
   FIRST(Primary) = FIRST(FunCall) ∪ FIRST(Id) ∪ FIRST( (Expr) )
   = {Id} ∪ {Id, (} = { Id, ( }
   FIRST(FunCall) = FIRST( Id ( OptArgs ) ) = {Id}
   FIRST(OptArgs) = FIRST(ε) ∪ FIRST(Args) = { • } ∪ ∅ = { • }
   FIRST(Args) = FIRST(Expr) ∪ FIRST(Args , Expr) = ∅ ∪ ∅ = ∅
3. (Listing only sets that change)
   FIRST(Expr) = FIRST(Expr + Term) ∪ FIRST(Term) = { Id, ( }
4. FIRST(Args) = FIRST(Expr) ∪ FIRST(Args , Expr) = { Id, ( }
5. FIRST(OptArgs) = FIRST(ε) ∪ FIRST(Args) = { • } ∪ { Id, ( } = { • , Id, ( }

For FOLLOW sets, we can calculate using the definition:

$$\begin{aligned} \text{FOLLOW}(A) = & \{ t \mid \exists \text{ production } B \text{ -> } \ldots A\alpha \text{ where } t \in \text{FIRST}(\alpha) \} & (1) \\ & \cup \{ \text{eof} \mid A \text{ is the start symbol} \} & (2) \\ & \cup \bigcup_{\mathbf{B}} \{ \text{FOLLOW}(B) \mid \exists \text{ production } B \text{ -> } \alpha A \} & (3) \end{aligned}$$

Again, start with FOLLOW(A) = ∅ for all non-terminals A, and iterate:

0. FOLLOW(Expr) = FOLLOW(Term) = ... = ∅
1. FOLLOW(Expr) = { eof, +, ) } (rules 1 & 2)
   FOLLOW(Term) = { * } (rule 1)
   FOLLOW(Primary) = ∅
   FOLLOW(FunCall) = ∅
   FOLLOW(OptArgs) = { ) } (rule 1)
   FOLLOW(Args) = { , } (rule 1)
2. FOLLOW(Expr) = {eof, +, ), ‘,’} (rules 1, 2, 3)
   FOLLOW(Term) = {*, eof, +, ) } (rules 1, 3)

FOLLOW(FunCall) = ∅
FOLLOW(OptArgs) = { ) } (rule 1)
FOLLOW(Args) = { ‘,’, ) } (rule 1, 3)

3. (Listing only sets that changed)
   FOLLOW(Primary) = { *, eof, +, ) } (rule 3)
   FOLLOW(FunCall) = { * } (rule 3)
4. FOLLOW(FunCall) = { *, eof, +, ) } (rule 3)