FIRST and FOLLOW sets

To compute FIRST(X) for all grammar symbols X, apply the following rules until no more terminals or ϵ can be added to any FIRST set.

1. If X is a terminal, then FIRST(X) = $\{X\}$
2. If X is a nonterminal, and FIRST(X) = $Y_1 Y_2 \ldots Y_n$ is a production rule, then:
   - Everything in FIRST(Y_1) is in FIRST(X)
   - If Y_1 is nullable, meaning the production $Y_1 \to \epsilon$ exists, then everything in FIRST(Y_2) is also in FIRST(X)
   - If both Y_1 and Y_2 are nullable, meaning both $Y_1 \to \epsilon$ and $Y_2 \to \epsilon$ exist, then everything in FIRST(Y_3) is in FIRST(X). Continue for all Y_n that can be nulled
3. If $X \to \epsilon$, then add ϵ to FIRST(X)

To compute FOLLOW(A) for all nonterminals A, apply the following rules until nothing can be added to any FOLLOW set:

- Place \$ in FOLLOW(S), where S is the start symbol, and \$ is the end-of-input marker
- If there is a production rule $X \to \alpha A \beta$, then everything in FIRST(β) except ϵ is in FOLLOW(A)
- If there is a production rule $X \to \alpha A$, or a production $X \to \alpha A \beta$ where β is nullable (contains $\beta \to \epsilon$), then everything in FOLLOW(X) is in FOLLOW(A)

For example, given our LL(1) grammar:

$S \to E$
$E \to TE'$
$E' \to +TE' \quad | \quad \epsilon$
$T \to FT'$
$T' \to *FT' \quad | \quad \epsilon$
$F \to (E) \quad | \quad \text{id}$

We can compute the FIRST sets as follows. We get the sets for F, T', and E' by directly inspecting their productions. We get the sets for S, E, and T by applying rule 2 above.

FIRST(S) = FIRST(E) = { (, id }
FIRST(E) = FIRST(T) = { (, id }
FIRST(E') = { +, ϵ }
FIRST(T) = FIRST(F) = { (, id }
FIRST(T') = { *, ϵ }
FIRST(F) = { (, id }

Once we have those, then we can compute the FOLLOW sets. For each, we identify each production the nonterminal appears in and include the relevant FIRST and FOLLOW sets as appropriate. For example, for FOLLOW(F) includes FIRST(T') because it appears directly after F in rules 4 and 5, but also includes FOLLOW(T) and FOLLOW(T') because T' is nullable in rules 4 and 5.

FOLLOW(S) = { \$ }
FOLLOW(E) = {) } ∪ FOLLOW(S) = {), \$ }
FOLLOW(E') = FOLLOW(E) = {), \$ }
FOLLOW(T) = FIRST(E') ∪ FOLLOW(E) ∪ FOLLOW(E')= { +,), \$ }
FOLLOW(T') = FOLLOW(T) = { +,), \$ }
FOLLOW(F) = FIRST(T') ∪ FOLLOW(T) ∪ FOLLOW(T') = { *, +,), \$ }

Constructing an LL(1) Parsing Table

The predictive parsing table $M[A, x]$ is generated using the FIRST and FOLLOW sets. The table has a row for each nonterminal in the grammar, and a column for each possible input symbol. The contents of each cell are a grammar production.

Remember that we are trying to parse an LL(1) grammar, which means that we only need to look at the leftmost symbol in the input string to know which production to apply, and that there can only be one valid sequence of productions to that symbol. Thus, we have two intiutions for how to build the parse table:

1. For each input symbol, we look at the FIRST sets of each nonterminal to see which ones can produce that input symbol.
2. For each nullable nonterminal, we look at their FOLLOW sets to see when we might apply the empty string (ϵ) production.

More formally: to generate the table, for each production rule $A \to \alpha$, do the following:

1. If α can derive a string starting with a terminal x (if x is in FIRST(A)), then add $A \to \alpha$ to $M[A, x]$
2. If α is nullable (can derive the empty string ϵ), then for all x that can follow A (x in FOLLOW(A)), add $A \to \alpha$ to $M[A, x]$
3. Any cell not assigned a production in this manner implicitly generates an **error**

Nonterminals S, E, T, and F are easiest, given that they only have one production each. We apply the first rule above- in each case, the FIRST set contains only (and \$, so we put the relevant productions into those cells.

For E', we first apply rule 1 from above, which gives the production in cell M[E',+]. We also notice that E' is nullable, so we apply the $E' \to \epsilon$ production for each terminal in FOLLOW(E'), which gives cells M[E',)] and M[E',\$].

Non-Terminal	Input Symbol					
	id	+	*	(	)	$
S	$S \to E$			$S \to E$		
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to$ id			$F \to (E)$		

For T', we again apply rule 1 from above, which gives the production in cell M[T',*]. We again notice that T' is nullable, so we apply the $T' \to \epsilon$ production for each terminal in FOLLOW(T'), which gives cells M[T',+], M[T',)], and M[T',$].

Table-driven LL(1) Parsing

Parsing can now be done with a stack machine, where the stack holds the current derivation. Initialize the stack with the start and the end-of-input symbol. At each step, we look at the leftmost nonterminal A in the derivation and the leftmost terminal x in the input string. We then look up cell M[A, x] in the parse table and apply the production there. If no rule exists in the table at that location, then there is a syntax error and the grammar does not accept the input string. For example, to parse the input string 1+2*3:

Stack	Input	Table Cell	Action	Match
$S$$	1+2*3$	M[S,id]	$S \to E$	
$E$$	1+2*3$	M[E,id]	$E \to TE'$	
$TE'$$	1+2*3$	M[T,id]	$T \to FT'$	
$FT'E'$$	1+2*3$	M[F, id]	$F \to$ id	
id$T'E'$$	1+2*3$		MATCH	id(1)
$T'E'$$	+2*3$	M[T',+]	$T' \to \epsilon$	
$E'$$	+2*3$	M[E',+]	$E' \to +TE'$	
$+TE'$$	+2*3$		MATCH	+
$TE'$$	2*3$	M[T,id]	$T \to FT'$	
$FT'E'$$	2*3$	M[F,id]	$F \to$ id	
id$T'E'$$	2*3$		MATCH	id(2)
$T'E'$$	*3$	M[T',*]	$T' \to *FT'$	
$*FT'E'$$	*3$		MATCH	*
$FT'E'$$	3$	M[F,id]	$F \to$ id	
id$T'E'$$	3$		MATCH	id(3)
$T'E'$$	$	M[T',$]	$T' \to \epsilon$	
$E'$$	$	M[E',$]	$E' \to \epsilon$	
$	$		MATCH	$