To compute the first and follow sets for use in make parse tables you must pretreat the grammar by removing alternations and then prefixing the grammar with a production that attaches an End of Input token (often denoted by a $) to start symbol:

\[ \text{Pgm} = \text{start}'$' \]

If we have productions of the form some with alternations:

\[ \text{A}_1 = X_1 X_2 X_3 \ldots X_{n_1}a \mid X_1 X_2 X_3 \ldots X_{n_1}b \]
\[ \text{A}_2 = X_1 X_2 X_3 \ldots X_{n_2} \]
\[ \text{A}_3 = X_1 X_2 X_3 \ldots X_{n_3}a \mid X_1 X_2 X_3 \ldots X_{n_3}b \]
\[ \ldots \]
\[ \text{A}_k = X_1 X_2 X_3 \ldots X_{n_k} \]

Then we convert to this equivalent grammar with no alternations:

Note the duplicates on the left hand side. Underscore denotes a subscript. \( X_{n_1} \) reads \( X \text{ sub } (n \text{ sub 1}) \).

\[ \text{A}_0 = \text{A}_1 \$
\[ \text{A}_1 = X_1 X_2 X_3 \ldots X_{n_1}a \\
\text{A}_1 = X_1 X_2 X_3 \ldots X_{n_1}b \\
\text{A}_2 = X_1 X_2 X_3 \ldots X_{n_2} \\
\text{A}_3 = X_1 X_2 X_3 \ldots X_{n_3}a \\
\text{A}_3 = X_1 X_2 X_3 \ldots X_{n_3}b \\
\ldots \\
\text{A}_k = X_1 X_2 X_3 \ldots X_{n_k} \]

The productions can be numbered \( P_1, \ldots P_m \).

Then we proceed with the algorithms.

We will begin by describing the algorithms finding the values of the arrays First and Follow. \( \text{First}(A) \) is an array indexed by a terminal or nonterminal and its value is a set of terminals and/or \( \text{\epsilon} \).
Follow(A) is an array indexed by a nonterminal and its value is a set of terminals and does not contain \( \epsilon \).

**COMPUTING First[A]**

-------------

// Procedure computeFirst

//
// Input: productions P_1, P_2, ..., P_m where P_i = A::=X_1 X_2 X_3 ...X_n
// with no alternations allowed in the productions.
//
// Output: Computes first of a term or nonterm accounting for nullability
// and multiple productions for the same nonterm.
//
// First is an array indexed by a terminal or
// nonterminal and its value is a set of terminals and/or \( \epsilon \).
//
// First[A] for nonterminal A is the set of all possible tokens that
// can occur as the first token of a sentence derived from A.
// First[A] for terminal A is simply the set \{ A \}.
//
// Compute the first sets for all tokens from productions P_1, P_2, ..., P_m
// where no production contains an alternation
//
// CALLS: computeFirstOfList(X_1, X_2, ... X_n)

procedure computeFirst({P_1, P_2, ...P_m}) // works on a list of productions
 // initial value for the First of anything
 foreach A \elemof TERMS do First[A] = \{A\}
 foreach A \elemof NONTERMS do First[A] = \{

 // loop until nothing new happens updating the First sets
 while stillchanging any First[A] do
    foreach production P_i = A::=X_1, X_2, ... X_n do
       First[A] <- First[A] \union computeFirstOfList(X_1, X_2, ... X_n)
    end foreach
 end while
 end

// Procedure computeFirstOfList

//
// Computes the First of a rhs rather than just a token!
//
// This computes the set of tokens that can occur as the first
// token of a sentence derived from this rhs (right hand side) of
// of production. That is X_1, X_2, ... X_n is a concatenation of
// terminals and nonterminals often found on the right hand side
// of a production. This is nontrivial because some of the leading
// nonterminals on the rhs can go to epsilon.
//
// REFS: First[X_i] (does not use Follow)
//
procedure computeFirstOfList(X_1, X_2, ... X_n)
    Tmp = {}
    k=0
do
    k++
    Tmp <- Tmp \ union First[X_k]-{\epsilon}
while k<n & \epsilon isin First[X_k]
// \epsilon only if X_1, X_2, ... X_n -> \epsilon
// Note: this test can only possibly work if k==n:
if \epsilon isin First[X_k] then Tmp <- Tmp \ union {\epsilon}
return Tmp
end

Note:

1. IMPORTANT: if grammar has no \epsilon then the procedure
   computeFirstOfList(X_1, X_2, ... X_n) simply returns First[X_1]

2. since \epsilon is removed when adding to First inside the do/while
   \epsilon can only appear when the entire argument list can be
   replaced by \epsilon (called this production is called NULLABLE).

3. First Sets can contain \epsilon as an element. Follow Sets cannot as we'll see.

4. Conceptually, computeFirst generates a relation of the form:
   First[A] = First[\alpha] \ union First[\beta] \ union ... \ union { \epsilon },
   for each production where A occurs on the left hand side (lhs).
   This is based on the next point.

5. Conceptually, computeFirstOfList generates a relation of the form:
   computeFirstOfList(X_1, X_2, ... X_n) = First[\alpha] \ union First[\beta]
   \ union ... \ union { \epsilon } where terms are added based on if
   all of the terms before it in the rhs are nullable.

COMPUTING Follow[A]
-------------------

// Procedure computeFollow
//
// Input: productions P_1, P_2, ..., P_m where P_i = A::=X_1 X_2 X_3 ...X_n
//     with no alternations allowed in the productions.
// Output: Follow is an array indexed by a nonterminal and its value
// is a set of terminals.

// Follow[A] is the set of all possible tokens that
// can occur after nonterminal A. This procedure assumes you
// have computed the First sets

// CALLS: computeFirstOfList(X_i+1,X_i+2...) which requires First
// REFS: Follow[]

procedure computeFollow({P_1, P_2, ...P_m}) // works on a list of productions
// initialize all the follow sets
foreach A \elemof NONTERMS do Follow[A] = {}
Follow[<start>]={$}

// loop until nothing new happens updating the Follow sets
while stillchanging any Follow[A] do {
    foreach P_i do {
        foreach X_i do // over elements in right hand side!
            if X_i \elemof NONTERMS then { // the body of this loop is over all the nonterms on the rhs
                Follow[X_i] <- Follow[X_i] \union computeFirstOfList(X_i+1 X_i+2 ...)
                if \epsilon \elemof computeFirstOfList(X_i+1,X_i+2...) then
                    Follow[X_i] <- Follow[X_i] \union Follow[A]
                end if
            end if
        end foreach
    end foreach
end while
end

Note:
1. that since \epsilon is subtracted from First[X_i+1 X_i+2 ...] before
   adding to Follow[X_i], \epsilon CANNOT OCCUR IN A FOLLOW SET. This is
   unlike the first set.

2. Only follow sets contain the end of input symbol '$'.

3. Conceptually, computeFollow generates a relation of the form:
   Follow[X_i] = First[\alpha] \union First[\beta]
   \union ... \union Follow[A] - { \epsilon } where any of these terms may be absent
   depending on the grammar. This is based on the next point.

PREDICT SET
-------------

The Predict Set of a production tells what lookahead tokens predict the
use of that production A::=X_1, X_2, ... X_n
This is simply computeFirstOfList but if that is empty then use Follow.

// compute the predict set of a production
procedure computePredict(A::=X_1, X_2, ... X_n)
    Tmp <- computeFirstOfList(X_1, X_2, ... X_n)
    if \epsilon \elemof Tmp then
        Tmp <- Tmp \union Follow[A] // only need Follow if there is epsilon in computeFirstOfList
    endif
    return Tmp - { \epsilon }
end

If there is NO \epsilon in the grammar:

procedure computePredict(A::=X_1, X_2, ... X_n)
    return First[X_1]
end

Summary:

Function | Uses          | TakesThisTypeOfArgument
-------- | ------------- | ------------------------
computeFirst | First,Follow | SetOfProductions
computeFollow | First,Follow | SetOfProductions
computeFirstOfList | First        | RHSofProduction
computePredict | First,Follow | Production

CONSTRUCTING THE LL PARSE TABLE
---------------------------------

If P_i are productions then we want
M(A, t) where A \elemof NONTERMS and t \elemof computePredict(P_i) should
contain a reference to production P_i as the action to take.

This means:

An LL(1) parse table can be built if for every pair of productions P_i, P_j with lhs(P_i) = lhs(P_j) that it is the case that
computePredict(P_i) intersect computePredict(P_j) = \emptyset

In other words:

there will be two productions in some M(A, t) which means
we don't know which to do in that case.
CAREFUL STEPS TO AUTOMATICALLY FINDING THE LL PARSE TABLE
-----------------------------------------------
1. remove the alternation and list the terms and nonterms
2. compute first sets for nonterminals
3. compute the follow sets (only needed if \epsilon is in grammar)
4. compute the predict sets.
5. create LL Parse Table

now you are ready to parse.

-----------------------------------------------
EXAMPLE 1: NO \epsilon EXAMPLE

Given the following grammar with 5 productions which include alternation

<exp> ::= <exp> <addop> <term> | <term>
<addop> ::= + | -
<term> ::= <term> <mulop> <factor> | <factor>
<mulop> ::= *
<factor> ::= ( <exp> ) | num

STEP 1: REMOVE ALTERNATIONS (accept for some terminals) and
list the terms and nonterms. This is done for clarity

list of productions without alternation:

1) <exp> ::= <exp> <addop> <term>
2) <exp> ::= <term>
3) <addop> ::= + | - <-- cheating here
4) <term> ::= <term> <mulop> <factor>
5) <term> ::= <factor>
6) <mulop> ::= *
7) <factor> ::= ( <exp> )
8) <factor> ::= num

TERMS = {+, -, *, (, ), num}
NONTERMS = {<exp>, <addop>, <term>, <mulop>, <factor>}

Important Observation:
* The discovery of the first sets will be driven by the nonterminals in the lhs of the productions.
* The discovery of the follow sets will be driven by the nonterminals in the rhs of the productions.

STEP 2: COMPUTE THE FIRST SET

One way to think of the computeFirst algorithm is that it sets up relationships

First[exp] = First[exp] \union First[term]

<table>
<thead>
<tr>
<th></th>
<th>pass 1</th>
<th>pass 2</th>
<th>pass 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;exp&gt;</td>
<td>First[exp], First[term]</td>
<td>First[term]</td>
<td>(,num</td>
</tr>
<tr>
<td>&lt;addop&gt;</td>
<td>+,-</td>
<td>+,-</td>
<td>+,-</td>
</tr>
<tr>
<td>&lt;term&gt;</td>
<td>First[term], First[factor]</td>
<td>First[factor]</td>
<td>(,num</td>
</tr>
<tr>
<td>&lt;mulop&gt;</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>&lt;factor&gt;</td>
<td>(,num</td>
<td>(,num</td>
<td>(,num</td>
</tr>
</tbody>
</table>

STEP 3: COMPUTE THE FOLLOW SET. Not really needed because there are no \(\epsilon\)s, but we do it here for practice. We will do this two ways. Note: only productions 1,2,4,5,7 affect the follow sets since the rhs of 3, 6, 8 are nothing but terminals.

1)  \( <\text{exp}> ::= <\text{exp}> <\text{addop}> <\text{term}> \)
2)  \( <\text{exp}> ::= <\text{term}> \)
4)  \( <\text{term}> ::= <\text{term}> <\text{mulop}> <\text{factor}> \)
5)  \( <\text{term}> ::= <\text{factor}> \)
7)  \( <\text{factor}> ::= ( <\text{exp}> ) \)

Because this grammar has no \(\epsilon\)s, more complex flow through the algorithm is avoided and the process is simply a collection of set dependencies. I will list out the dependencies and fill in the data in the three steps below. \(\text{fst}\) stands for First and \(\text{fol}\) for Follow to save room. The first line below essentially means Follow[exp] = First[addop] \union First[" "].

Let’s sketch out what will happen.
For simplicity, just list out the relationships from each production:

<table>
<thead>
<tr>
<th></th>
<th>prod 1</th>
<th>prod 2</th>
<th>prod 4</th>
<th>prod 5</th>
<th>prod 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;exp&gt;</td>
<td>(,num</td>
<td>(\text{fst}(\text{addop}))</td>
<td>(\text{fst}(&quot;)&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;addop&gt;</td>
<td>+,-</td>
<td>(\text{fst}(\text{term}))</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;term&gt;</td>
<td>(,num</td>
<td>(\text{fol}(\text{exp}))</td>
<td>(\text{fol}(\text{exp}))</td>
<td>(\text{fst}(\text{mulop}))</td>
<td></td>
</tr>
<tr>
<td>&lt;mulop&gt;</td>
<td>*</td>
<td></td>
<td></td>
<td>(\text{fst}(\text{factor}))</td>
<td></td>
</tr>
<tr>
<td>&lt;factor&gt;</td>
<td>(,num</td>
<td></td>
<td></td>
<td>(\text{fol}(\text{term}))</td>
<td>(\text{fol}(\text{term}))</td>
</tr>
</tbody>
</table>
IMPORTANT: Initialize fol(exp) = $

First[] replaces the First sets AND each row represents the follow set. e.g.
Follow[<exp>] = {$(,+,-,)} initially below:

<table>
<thead>
<tr>
<th>Production</th>
<th>Prod 1</th>
<th>Prod 2</th>
<th>Prod 4</th>
<th>Prod 5</th>
<th>Prod 7</th>
<th>Follow</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;exp&gt;</td>
<td>(,num</td>
<td>+,-</td>
<td></td>
<td></td>
<td></td>
<td>$,+,-,)</td>
</tr>
<tr>
<td>addop</td>
<td>+,-</td>
<td>(,num</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>term</td>
<td>(,num</td>
<td>fol(exp)</td>
<td>fol(exp)</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mulop</td>
<td>*</td>
<td>(,num</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>factor</td>
<td>(,num</td>
<td>fol(term)</td>
<td>fol(term)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Then iterate over the follow sets:

<table>
<thead>
<tr>
<th>Production</th>
<th>Prod 1</th>
<th>Prod 2</th>
<th>Prod 4</th>
<th>Prod 5</th>
<th>Prod 7</th>
<th>Follow</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;exp&gt;</td>
<td>(,num</td>
<td>+,-</td>
<td></td>
<td></td>
<td></td>
<td>$,+,-,)</td>
</tr>
<tr>
<td>addop</td>
<td>+,-</td>
<td>(,num</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>term</td>
<td>(,num</td>
<td>$,+,-,)</td>
<td>$,+,-,</td>
<td>*</td>
<td></td>
<td>$,+,-,*</td>
</tr>
<tr>
<td>mulop</td>
<td>*</td>
<td>(,num</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>factor</td>
<td>(,num</td>
<td>$,+,-,)</td>
<td>$,+,-,</td>
<td>$,+,-,*</td>
<td>$,+,-,*</td>
<td></td>
</tr>
</tbody>
</table>

Let's see that again only this time we compute the Follow set by running through the algorithm. If a first or follow set is mentioned here it stands for the empty set (not the empty string) and is just there to be informative about what information we are using.

1) <exp> ::= <exp> <addop> <term>
2) <exp> ::= <term>
4) <term> ::= <term> <mulop> <factor>
5) <term> ::= <factor>
7) <factor> ::= ( <exp> )

<table>
<thead>
<tr>
<th>Production</th>
<th>Prod 1</th>
<th>Prod 2</th>
<th>Prod 4</th>
<th>Prod 5</th>
<th>Prod 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;exp&gt;</td>
<td>(,num</td>
<td>fst(addop)</td>
<td></td>
<td>fst(&quot;&quot;))</td>
<td></td>
</tr>
<tr>
<td>addop</td>
<td>+,-</td>
<td>fst(term)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>term</td>
<td>(,num</td>
<td>fol(exp)</td>
<td>fol(exp)</td>
<td>fst(mulop)</td>
<td></td>
</tr>
<tr>
<td>mulop</td>
<td>*</td>
<td></td>
<td></td>
<td>fst(factor)</td>
<td></td>
</tr>
<tr>
<td>factor</td>
<td>(,num</td>
<td>fol(term)</td>
<td>fol(term)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

pass 0:
Initialize fol(exp) = $

Pass 1:
Under each prod is what is ADDED for the production given at the top of the column. The TOTAL column is what is in each follow set at the
end of the pass. Order of evaluation in each nonterminal is determined by
the order of the productions (from prod 1 to prod 7).

<table>
<thead>
<tr>
<th></th>
<th>First</th>
<th>prod 1</th>
<th>prod 2</th>
<th>prod 4</th>
<th>prod 5</th>
<th>prod 7</th>
<th>Follow</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;exp&gt;</td>
<td>(,num $,+,-$)</td>
<td>$,+,-,$</td>
<td>$,+,-,$</td>
<td>$,+,-,$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;addop&gt;</td>
<td>+,-</td>
<td>(,num )</td>
<td>$,+,-,$</td>
<td>$,+,-,$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;term&gt;</td>
<td>(,num $,+,-$ $,+,-$ *</td>
<td>$,+,-,*$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;mulop&gt;</td>
<td>*</td>
<td>(,num )</td>
<td>$,+,-,*$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;factor&gt;</td>
<td>(,num $,+,-,<em>$ $,+,-,</em>$ $,+,-,*$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pass 2:

<table>
<thead>
<tr>
<th></th>
<th>First</th>
<th>prod 1</th>
<th>prod 2</th>
<th>prod 4</th>
<th>prod 5</th>
<th>prod 7</th>
<th>Follow</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;exp&gt;</td>
<td>(,num $,+,-$)</td>
<td>$,+,-,$</td>
<td>$,+,-,$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;addop&gt;</td>
<td>+,-</td>
<td>(,num )</td>
<td>$,+,-,$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;term&gt;</td>
<td>(,num $,+,-$ $,+,-$ *</td>
<td>$,+,-,*,$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;mulop&gt;</td>
<td>*</td>
<td>(,num )</td>
<td>$,+,-,*$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;factor&gt;</td>
<td>(,num $,+,-,<em>$ $,+,-,</em>$ $,+,-,*$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pass 3: no change

STEP 4. Compute the predict sets for each production. In this case
it is essentially the first set of the first symbol on the right
hand side:

1) <exp> ::= <exp> <addop> <term> (,num
2) <exp> ::= <term> (,num
3) <addop> ::= + | - +,-
4) <term> ::= <term> <mulop> <factor> (,num
5) <term> ::= <factor> (,num
6) <mulop> ::= * *
7) <factor> ::= ( <exp> ) (,
8) <factor> ::= num num

The sets on the LEFT for an expression on the RIGHT must
have empty intersections.

STEP 5: compute the LL parse table.

Remember this is a top-down parser so given the nonterm on left of
table and terminal at top of table what production should we use which
is M(A, t) in the table.

M(A, t) where A \in NONTERMS and t \in computePredict(P_i)

M $ + - * ( ) num
<exp>    stop     1,2   1,2
<addop>   3     3
<term>    4,5   4,5
<mulop>   6
<factor>  7     8

Note:
1. since no \( \epsilon \)s are present Follow sets are not needed but it was good practice. The next example is more complicated.

IMPORTANT:
Because \( M(<exp>, '(') \) can be a 1 OR 2 the machine is not well defined!!! This means that the grammar we gave is NOT an LL(1) grammar. The problem that the LL(1) parser is suffering is the same one we had with recursive descent parsing. We can fix this. Let's look at an example of the fix before we generalize.

EXAMPLE 2: an example with \( \epsilon \) in the grammar

Take the same grammar as in example 1:

\[
\begin{align*}
<exp> & ::= <exp> <addop> <term> | <term> \\
<addop> & ::= + | - \\
<term> & ::= <term> <mulop> <factor> | <factor> \\
<mulop> & ::= * \\
<factor> & ::= ( <exp> ) | num
\end{align*}
\]

We will begin by removing left recursion as in section 4.2.3 creating two new nonterminals: expx and termx.

Step 0: remove left recursion

\[
\begin{align*}
<exp> & ::= <term> <expx> \\
<expx> & ::= <addop> <term> <expx> | \epsilon \\
<addop> & ::= + | - \\
<term> & ::= <factor> <termx> \\
<termx> & ::= <mulop> <factor> <termx> | \epsilon \\
<mulop> & ::= * \\
<factor> & ::= ( <exp> ) | num
\end{align*}
\]

STEP 1: remove alternation and list the terms and nonterms for clarity

0) \( <start> ::= <exp> $ \)
1) \( \langle \text{exp} \rangle ::= \langle \text{term} \rangle \ \langle \text{expx} \rangle \)

2) \( \langle \text{expx} \rangle ::= \langle \text{addop} \rangle \ \langle \text{term} \rangle \ \langle \text{expx} \rangle \)

3) \( \langle \text{expx} \rangle ::= \epsilon \)

4) \( \langle \text{addop} \rangle ::= + \ | - \)

5) \( \langle \text{term} \rangle ::= \langle \text{factor} \rangle \ \langle \text{termx} \rangle \)

6) \( \langle \text{termx} \rangle ::= \langle \text{mulop} \rangle \ \langle \text{factor} \rangle \ \langle \text{termx} \rangle \)

7) \( \langle \text{termx} \rangle ::= \epsilon \)

8) \( \langle \text{mulop} \rangle ::= * \)

9) \( \langle \text{factor} \rangle ::= ( \ \langle \text{exp} \ \rangle ) \)

a) \( \langle \text{factor} \rangle ::= \text{num} \)

TERMS = \{+, -, *, (, ), num\}

NONTERMS = \{\langle \text{exp} \rangle, \langle \text{expx} \rangle, \langle \text{addop} \rangle, \langle \text{term} \rangle, \langle \text{termx} \rangle, \langle \text{mulop} \rangle, \langle \text{factor} \rangle\} 

STEP 2: compute the first set

<table>
<thead>
<tr>
<th>Production</th>
<th>Pass 1</th>
<th>Pass 2</th>
<th>Pass 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>\langle \text{start} \rangle ::= \langle \text{exp} \rangle $</td>
<td>\text{first(exp)}</td>
<td>\text{first(exp)}</td>
<td>(,num</td>
</tr>
<tr>
<td>\langle \text{exp} \rangle ::= \langle \text{term} \rangle \ \langle \text{expx} \rangle</td>
<td>\text{first(term)}</td>
<td>\text{first(factor)}</td>
<td>(,num</td>
</tr>
<tr>
<td>\langle \text{expx} \rangle ::= \langle \text{addop} \rangle \ \langle \text{term} \rangle \ \langle \text{expx} \rangle</td>
<td>+,-,\epsilon</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\langle \text{expx} \rangle ::= \epsilon</td>
<td>+,-,\epsilon</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\langle \text{addop} \rangle ::= +,-</td>
<td>+,-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\langle \text{term} \rangle ::= \langle \text{factor} \rangle \ \langle \text{termx} \rangle</td>
<td>(,num</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\langle \text{termx} \rangle ::= \langle \text{mulop} \rangle \ \langle \text{factor} \rangle \ \langle \text{termx} \rangle</td>
<td>*,\epsilon</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\langle \text{termx} \rangle ::= \epsilon</td>
<td>*,\epsilon</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\langle \text{mulop} \rangle ::= *</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\langle \text{factor} \rangle ::= ( \ \langle \text{exp} \ \rangle )</td>
<td>(,num</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: \( \epsilon \) occurs only where the nonterminal can disappear by application of a production.

STEP 3: compute the follow set

Note: only productions 0,1,2,5,6,9 affect the follow sets (ignoring nullable NTs)

0) \( \langle \text{start} \rangle ::= \langle \text{exp} \rangle $ 

1) \( \langle \text{exp} \rangle ::= \langle \text{term} \rangle \ \langle \text{expx} \rangle 

2) \( \langle \text{expx} \rangle ::= \langle \text{addop} \rangle \ \langle \text{term} \rangle \ \langle \text{expx} \rangle 

3) \( \langle \text{expx} \rangle ::= \epsilon 

5) \( \langle \text{term} \rangle ::= \langle \text{factor} \rangle \ \langle \text{termx} \rangle 

6) \( \langle \text{termx} \rangle ::= \langle \text{mulop} \rangle \ \langle \text{factor} \rangle \ \langle \text{termx} \rangle 

7) \( \langle \text{termx} \rangle ::= \epsilon 

9) \( \langle \text{factor} \rangle ::= ( \ \langle \text{exp} \ \rangle ) 

The following shows what happens to the follow sets as each production is analyzed. Note for example that we add in fol(exp) for term because in prod 1 \( \langle \text{expx} \rangle \) can go to epsilon in production 3!

Account for each production:
Group by nonTerminal. Remember Follow sets do not have $\epsilon$-s

\[
\begin{align*}
\text{First} & \quad \text{prod 0&1} \quad \text{prod 2} \quad \text{prod 5} \quad \text{prod 6} \quad \text{prod 9} \\
<\text{start}> & \quad (,\text{num} \quad $) \\
<\text{exp} & \quad (,\text{num} \quad $) \\
<\text{expx} & \quad +,-,\epsilon \quad \text{fol}(\text{exp}) \quad \text{fol}(\text{expx}) \\
<\text{addop} & \quad +,- \quad \text{fst}(\text{term}) \\
<\text{term} & \quad (,\text{num} \quad \text{fst}(\text{expx}) \quad \text{fst}(\text{expx}) \\
& \quad \text{& fol}(\text{exp}) \quad \text{& fol}(\text{expx}) \\
<\text{termx} & \quad *,\epsilon \quad \text{fol}(\text{term}) \quad \text{fol}(\text{termx}) \\
<\text{mulop} & \quad * \quad \text{& fol}(\text{factor}) \\
<\text{factor} & \quad (,\text{num} \quad \text{fst}(\text{termx}) \quad \text{fst}(\text{termx}) \\
& \quad \text{& fol}(\text{term}) \quad \text{& fol}(\text{termx})
\end{align*}
\]

pass 0:

\[
\text{Initialize fol}(\text{exp}) = $
\]

\[
\begin{align*}
\text{First} & \quad \text{Follow} \\
<\text{start}> & \quad (,\text{num} \quad $) \\
<\text{exp} & \quad (,\text{num} \quad $) \\
<\text{expx} & \quad +,-,\epsilon \quad \text{fol}(\text{exp}) \\
<\text{addop} & \quad +,- \quad (,\text{num} \\
<\text{term} & \quad (,\text{num} \quad +,- \quad \text{& fol}(\text{exp}) \quad \text{& fol}(\text{expx}) \\
<\text{termx} & \quad *,\epsilon \quad \text{fol}(\text{term}) \quad \text{& fol}(\text{termx}) \\
<\text{mulop} & \quad * \quad (,\text{num} \\
<\text{factor} & \quad (,\text{num} \quad * \quad \text{& fol}(\text{term}) \quad \text{& fol}(\text{termx})
\end{align*}
\]

\[
\begin{align*}
\text{First} & \quad \text{Follow} \\
<\text{start}> & \quad (,\text{num} \quad $) \\
<\text{exp} & \quad (,\text{num} \quad $) \\
<\text{expx} & \quad +,-,\epsilon \quad $) \\
<\text{addop} & \quad +,- \quad (,\text{num} \\
<\text{term} & \quad (,\text{num} \quad +,-,\text{$,$}) \\
<\text{termx} & \quad *,\epsilon \quad +,-,\text{$,$) \\
<\text{mulop} & \quad * \quad (,\text{num} \\
<\text{factor} & \quad (,\text{num} \quad *,+,-,\text{$,$})
\end{align*}
\]
STEP 4. Compute the predict sets:

<table>
<thead>
<tr>
<th>Production</th>
<th>Predict Set</th>
<th>Predict Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) ( &lt;exp&gt; ::= &lt;term&gt; &lt;expx&gt; )</td>
<td>First[( \text{term} )] (,num</td>
<td>(,num</td>
</tr>
<tr>
<td>2) ( &lt;expx&gt; ::= &lt;addop&gt; &lt;term&gt; &lt;expx&gt; )</td>
<td>First[( \text{addop} )] +,-</td>
<td>+,-</td>
</tr>
<tr>
<td>3) ( &lt;expx&gt; ::= \epsilon )</td>
<td>Follow[( \text{expx} )] $,$)</td>
<td>$,$)</td>
</tr>
<tr>
<td>4) ( &lt;addop&gt; ::= +</td>
<td>- )</td>
<td>First[( +</td>
</tr>
<tr>
<td>5) ( &lt;term&gt; ::= &lt;factor&gt; &lt;termx&gt; )</td>
<td>First[( \text{factor} )] (,num</td>
<td>(,num</td>
</tr>
<tr>
<td>6) ( &lt;termx&gt; ::= &lt;mulop&gt; &lt;factor&gt; &lt;termx&gt; )</td>
<td>First[( \text{mulop} )] *</td>
<td>*</td>
</tr>
<tr>
<td>7) ( &lt;termx&gt; ::= \epsilon )</td>
<td>Follow[( \text{termx} )] +,-,$,)</td>
<td>+,-,$,)</td>
</tr>
<tr>
<td>8) ( &lt;mulop&gt; ::= \star )</td>
<td>First[( \star )] *</td>
<td>*</td>
</tr>
<tr>
<td>9) ( &lt;factor&gt; ::= ( &lt;exp&gt; ) )</td>
<td>First[( ()] (</td>
<td>(</td>
</tr>
<tr>
<td>a) ( &lt;factor&gt; ::= \text{num} )</td>
<td>First[( \text{num} )] num</td>
<td>num</td>
</tr>
</tbody>
</table>

STEP 5. Create \( \text{M(NONTERMS, TERMS)} \)

\( \text{M(A, t)} \) where \( \text{A} \in \text{\&\in\text{NONTERMS}} \) and \( \text{t} \in \text{\&\in\text{computePredict(A::=X_1 X_2...X_n)}} \)

<table>
<thead>
<tr>
<th>M</th>
<th>$</th>
<th>+</th>
<th>-</th>
<th>*</th>
<th>( )</th>
<th>num</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;exp&gt;</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;expx&gt;</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;addop&gt;</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;term&gt;</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;termx&gt;</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>&lt;mulop&gt;</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;factor&gt;</td>
<td>9</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RUN THE EXAMPLE ON SOME INPUT

--------------

PARSE STACK | INPUT | production |
-------------|-------|-----------|
\( \text{exp}$ | $3+4*5$ | 1 |
\( <\text{term}> <\text{expx}> $ | $3+4*5$ | 5 |
\( <\text{factor}> <\text{termx}> <\text{expx}> $ | $3+4*5$ | a |
\( \text{num}<\text{termx}> <\text{expx}> $ | $3+4*5$ | 13 |
Holy cow! It works!

Postscript:

A grammar is LL(1) if:
1. For every production  \( A ::= a_1 | a_2 | \ldots \)
   \( \forall i, j \ i \neq j \ \text{First}[a_i] \ \text{intersect} \ \text{First}[a_j] \ \text{is empty} \)
   AND
2. if \( \epsilon \ \text{elemof} \ \text{First}[A] \) then \( \text{First}[A] \ \text{intersect} \ \text{Follow}[A] \ \text{is empty} \)