LL(1) parsing
A **top-down** parsing algorithm parses an input string in such a way that the implied traversal of the parse tree occurs from the root to the leaves.
Top-down parsing

- A **top-down** parsing algorithm parses an input string in such a way that the implied traversal of the parse tree occurs from the root to the leaves.

- Top-down parsers come in two forms: **backtracking parsers** and **predictive parsers**.

**LL(1) parsing**
A **top-down** parsing algorithm parses an input string in such a way that the implied traversal of the parse tree occurs from the root to the leaves.

Top-down parsers come in two forms: **backtracking parsers** and **predictive parsers**.

A predictive parser attempts to predict the next construction using one or more lookahead tokens.
A top-down parsing algorithm parses an input string in such a way that the implied traversal of the parse tree occurs from the root to the leaves.

Top-down parsers come in two forms: backtracking parsers and predictive parsers.

A predictive parser attempts to predict the next construction using one or more lookahead tokens.

Two well-known top-down parsing methods are recursive-descent parsing and LL(1) parsing.
Top-down parsing

- A **top-down** parsing algorithm parses an input string in such a way that the implied traversal of the parse tree occurs from the root to the leaves.

- Top-down parsers come in two forms: **backtracking parsers** and **predictive parsers**.

- A predictive parser attempts to predict the next construction using one or more lookahead tokens.

- Two well-known top-down parsing methods are **recursive-descent parsing** and **LL(1) parsing**.

- Recursive descent parsing is the most suitable method for a handwritten parser.
The first “L” in LL(1) refers to the fact that the input is processed from left to right.
The first “L” in LL(1) refers to the fact that the input is processed from left to right.

The second “L” refers to the fact that LL(1) parsing determines a leftmost derivation for the input string.
The first “L” in LL(1) refers to the fact that the input is processed from left to right.

The second “L” refers to the fact that LL(1) parsing determines a leftmost derivation for the input string.

The “1” in parentheses implies that LL(1) parsing uses only one symbol of input to predict the next grammar rule that should be used.
We start by considering the following grammar that generates strings of balanced parenthesis:

$$ S \rightarrow ( S ) S \mid \varepsilon $$
We start by considering the following grammar that generates strings of balanced parenthesis:

\[
S \rightarrow ( \ S \ ) \ S \mid \epsilon
\]

We will assume that $ marks the bottom of a stack and the end of input.
We start by considering the following grammar that generates strings of balanced parenthesis:
\[ S \rightarrow (S)S \mid \varepsilon \]
We will assume that $ marks the bottom of a stack and the end of input.

Parsing action of an LL(1) parser:

<table>
<thead>
<tr>
<th>Parsing Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 $S$</td>
<td>()$</td>
<td>$S \rightarrow (S)S$</td>
</tr>
<tr>
<td>2 $S)$</td>
<td>()$</td>
<td>match</td>
</tr>
<tr>
<td>3 $S$</td>
<td>)$</td>
<td>$S \rightarrow \varepsilon$</td>
</tr>
<tr>
<td>4 $S)$</td>
<td>)$</td>
<td>match</td>
</tr>
<tr>
<td>5 $S$</td>
<td>$</td>
<td>$S \rightarrow \varepsilon$</td>
</tr>
<tr>
<td>6 $S$</td>
<td>$</td>
<td>accept</td>
</tr>
</tbody>
</table>
We start by considering the following grammar that generates strings of balanced parenthesis:

\[ S \rightarrow ( S ) S | \varepsilon \]

We will assume that $ marks the bottom of a stack and the end of input.

<table>
<thead>
<tr>
<th>Parsing Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>()$</td>
<td>$S \rightarrow (S)S$</td>
</tr>
<tr>
<td>$(S)S$</td>
<td>()$</td>
<td>match</td>
</tr>
<tr>
<td>$(S)$</td>
<td>)$</td>
<td>$S \rightarrow \varepsilon$</td>
</tr>
<tr>
<td>$(S)S$</td>
<td>)$</td>
<td>match</td>
</tr>
<tr>
<td>$S$</td>
<td>$</td>
<td>$S \rightarrow \varepsilon$</td>
</tr>
<tr>
<td>$S$</td>
<td>accept</td>
<td></td>
</tr>
</tbody>
</table>

The \( LL(1) \) parsing table \( M[N, T] \):

\[
\begin{array}{ccc}
S & M[N, T] & (S)S \\
S & S \rightarrow (S)S & S \rightarrow \varepsilon \\
S & S \rightarrow \varepsilon & S \rightarrow \varepsilon
\end{array}
\]
LL(1) parsing tables

- We use the parsing table to decide which decision should be made if a given nonterminal $N$ is at the top of the parsing stack, based on the current input symbol $T$.

  1. If $A \rightarrow \alpha$ is a production choice, and there is a derivation $\alpha \Rightarrow^{*} a \beta$, where $a$ is a token, then we add $A \rightarrow \alpha$ to the table entry $M[A, a]$.

  2. If $A \rightarrow \alpha$ is a production choice, and there are derivations $\alpha \Rightarrow^{*} \varepsilon$ and $S \Rightarrow^{*} \beta A a \gamma$, where $S$ is the start symbol and $a$ is a token (or $\$$), then we add $A \rightarrow \alpha$ to the table entry $M[A, a]$.
We use the parsing table to decide which decision should be made if a given nonterminal $N$ is at the top of the parsing stack, based on the current input symbol $T$.

We add production choices to the LL(1) parsing table as follows:

1. If $A \rightarrow \alpha$ is a production choice, and there is a derivation $\alpha \Rightarrow ^* a \beta$, where $a$ is a token, then we add $A \rightarrow \alpha$ to the table entry $M[A, a]$.

2. If $A \rightarrow \alpha$ is a production choice, and there are derivations $\alpha \Rightarrow ^* \epsilon$ and $S \Rightarrow ^* \beta Aa \gamma$, where $S$ is the start symbol and $a$ is a token (or $\$$), then we add $A \rightarrow \alpha$ to the table entry $M[A, a]$.
We use the parsing table to decide which decision should be made if a given nonterminal $N$ is at the top of the parsing stack, based on the current input symbol $T$.

We add production choices to the LL(1) parsing table as follows:

1. If $A \rightarrow \alpha$ is a production choice, and there is a derivation $\alpha \Rightarrow^* a \beta$, where $a$ is a token, then we add $A \rightarrow \alpha$ to the table entry $M[A, a]$. 
We use the parsing table to decide which decision should be made if a given nonterminal $N$ is at the top of the parsing stack, based on the current input symbol $T$.

We add production choices to the LL(1) parsing table as follows:

1. If $A \rightarrow \alpha$ is a production choice, and there is a derivation $\alpha \Rightarrow^* a\beta$, where $a$ is a token, then we add $A \rightarrow \alpha$ to the table entry $M[A, a]$.

2. If $A \rightarrow \alpha$ is a production choice, and there are derivations $\alpha \Rightarrow^* \varepsilon$ and $S$\$ $\Rightarrow^* \beta A a \gamma$, where $S$ is the start symbol and $a$ is a token (or $\$$), then we add $A \rightarrow \alpha$ to the table entry $M[A, a]$.
The idea behind these rules are as follows:

1. In rule 1, given a token $a$ in the input, we wish to select a rule $A \rightarrow \alpha$ if $\alpha$ can produce an $a$ for matching.

2. In rule 2, if $A$ derives the empty string (via $A \rightarrow \alpha$), and if $a$ is a token that can legally come after $A$ in a derivation, then we want to select $A \rightarrow \alpha$ to make $A$ disappear.

These rules are difficult to implement directly, so we will develop algorithms involving first and follow sets.

A grammar is an LL(1) grammar if the associated LL(1) parsing table has at most one production in each table entry.
The idea behind these rules are as follows:

1. In rule 1, given a token $a$ in the input, we wish to select a rule $A \rightarrow \alpha$ if $\alpha$ can produce an $a$ for matching.
The idea behind these rules are as follows:

1. In rule 1, given a token $a$ in the input, we wish to select a rule $A \rightarrow \alpha$ if $\alpha$ can produce an $a$ for matching.
2. In rule 2, if $A$ derives the empty string (via $A \rightarrow \alpha$), and if $a$ is a token that can legally come after $A$ in a derivation, then we want to select $A \rightarrow \alpha$ to make $A$ disappear.
The idea behind these rules are as follows:

1. In rule 1, given a token $a$ in the input, we wish to select a rule $A \rightarrow \alpha$ if $\alpha$ can produce an $a$ for matching.

2. In rule 2, if $A$ derives the empty string (via $A \rightarrow \alpha$), and if $a$ is a token that can legally come after $A$ in a derivation, then we want to select $A \rightarrow \alpha$ to make $A$ disappear.

These rules are difficult to implement directly, so we will develop algorithms involving \textbf{first} and \textbf{follow sets}. 
The idea behind these rules are as follows:

1. In rule 1, given a token $a$ in the input, we wish to select a rule $A \rightarrow \alpha$ if $\alpha$ can produce an $a$ for matching.
2. In rule 2, if $A$ derives the empty string (via $A \rightarrow \alpha$), and if $a$ is a token that can legally come after $A$ in a derivation, then we want to select $A \rightarrow \alpha$ to make $A$ disappear.

These rules are difficult to implement directly, so we will develop algorithms involving first and follow sets.

A grammar is an **LL(1) grammar** if the associated LL(1) parsing table has at most one production in each table entry.
Before we give the precise definitions of First and Follow Sets, we show how to use it in the construction of LL(1) parsing tables.
Before we give the precise definitions of First and Follow Sets, we show how to use it in the construction of LL(1) parsing tables.

Repeat the following two steps for each nonterminal $A$ and each production $A \rightarrow \alpha$:
Before we give the precise definitions of First and Follow Sets, we show how to use it in the construction of LL(1) parsing tables.

Repeat the following two steps for each nonterminal $A$ and each production $A \rightarrow \alpha$:

1. For each token $a$ in First($\alpha$), add $A \rightarrow \alpha$ to the entry $M[A, a]$. 
Before we give the precise definitions of First and Follow Sets, we show how to use it in the construction of LL(1) parsing tables.

Repeat the following two steps for each nonterminal $A$ and each production $A \rightarrow \alpha$:

1. For each token $a$ in $\text{First}(\alpha)$, add $A \rightarrow \alpha$ to the entry $M[A, a]$.
2. If $\varepsilon$ is in $\text{First}(\alpha)$, for each element $a$ of $\text{Follow}(A)$ (where $a$ is a token or $a$ is $\$$), add $A \rightarrow \alpha$ to $M[A, a]$. 

LL(1) parsing
Now consider the grammar $S \rightarrow (S)S | \varepsilon$

Thus we get the following LL(1) parsing table:

\[
\begin{array}{c|cc}
  & ( & ) \\
\hline
S & S & \rightarrow (S)S \\
S & \varepsilon & \rightarrow \varepsilon
\end{array}
\]
Now consider the grammar $S \rightarrow (S)S | \varepsilon$

In this case we have that
\[
\begin{align*}
\text{First}( (S)S ) &= \{ ( \} \\
\text{First}(\varepsilon) &= \{ \varepsilon \} \\
\text{Follow}(S) &= \{ ), \$, $ \}
\end{align*}
\]
Now consider the grammar $S \rightarrow (S)S \mid \varepsilon$

In this case we have that

$\text{First}( (S)S ) = \{ ( \} \}

\text{First}(\varepsilon) = \{ \varepsilon \}

\text{Follow}(S) = \{ ),$ \}

Thus we get the following LL(1) parsing table:

<table>
<thead>
<tr>
<th>$M[N, T]$</th>
<th>( )</th>
<th>$$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>$S \rightarrow (S)S$</td>
<td>$S \rightarrow \varepsilon$</td>
</tr>
</tbody>
</table>
Let $X$ be a grammar symbol (a terminal or nonterminal) or $\varepsilon$. Then the set $\text{First}(X)$ consisting of terminals, and possibly $\varepsilon$, is defined as follows:

1. If $X$ is a terminal or $\varepsilon$, $\text{First}(X) = \{X\}$.
2. If $X$ is a nonterminal, then for each production choice $X \rightarrow X_1 X_2 \ldots X_n$, $\text{First}(X)$ contains $\text{First}(X_1) - \{\varepsilon\}$. If also for some $i < n$, all the sets $\text{First}(X_1), \ldots, \text{First}(X_i)$ contains $\varepsilon$, then $\text{First}(X)$ contains $\text{First}(X_{i+1}) - \{\varepsilon\}$. If all the sets $\text{First}(X_1), \ldots, \text{First}(X_n)$ contains $\varepsilon$, then $\text{First}(X)$ also contains $\varepsilon$. 

LL(1) parsing
Let $X$ be a grammar symbol (a terminal or nonterminal) or $\varepsilon$. Then the set $\text{First}(X)$ consisting of terminals, and possibly $\varepsilon$, is defined as follows:

1. If $X$ is a terminal or $\varepsilon$, $\text{First}(X) = \{X\}$. 

LL(1) parsing
Let $X$ be a grammar symbol (a terminal or nonterminal) or $\varepsilon$. Then the set $\text{First}(X)$ consisting of terminals, and possibly $\varepsilon$, is defined as follows:

1. If $X$ is a terminal or $\varepsilon$, $\text{First}(X) = \{X\}$.
2. If $X$ is a nonterminal, then for each production choice $X \rightarrow X_1X_2...X_n$, $\text{First}(X)$ contains $\text{First}(X_1) - \{\varepsilon\}$. If also for some $i < n$, all the sets $\text{First}(X_1),...,\text{First}(X_i)$ contains $\varepsilon$, then $\text{First}(X)$ contains $\text{First}(X_{i+1}) - \{\varepsilon\}$. If all the sets $\text{First}(X_1),...,\text{First}(X_n)$ contains $\varepsilon$, then $\text{First}(X)$ also contains $\varepsilon$. 

LL(1) parsing
We now define \textbf{First}(\alpha) for any string \( \alpha = X_1X_2...X_n \) (a string of terminals and nonterminals) as follows:

1. \textbf{First}(\alpha) contains \textbf{First}(X_1) \setminus \{\varepsilon\}.
2. For each \( i = 2, ..., n \), if \textbf{First}(X_k) contains \varepsilon for all \( k = 1, ..., i - 1 \), then \textbf{First}(\alpha) contains \textbf{First}(X_i) \setminus \{\varepsilon\}.
3. Finally, if for all \( i = 1, ..., n \), \textbf{First}(X_i) contains \varepsilon, then \textbf{First}(\alpha) contains \varepsilon.
We now define $\text{First}(\alpha)$ for any string $\alpha = X_1X_2...X_n$ (a string of terminals and nonterminals) as follows:

1. $\text{First}(\alpha)$ contains $\text{First}(X_1) - \{\varepsilon\}$. 

LL(1) parsing
We now define $\text{First}(\alpha)$ for any string $\alpha = X_1X_2...X_n$ (a string of terminals and nonterminals) as follows:

1. $\text{First}(\alpha)$ contains $\text{First}(X_1) - \{\varepsilon\}$.
2. For each $i = 2, ..., n$, if $\text{First}(X_k)$ contains $\varepsilon$ for all $k = 1, ..., i - 1$, then $\text{First}(\alpha)$ contains $\text{First}(X_i) - \{\varepsilon\}$. 
We now define $\text{First}(\alpha)$ for any string $\alpha = X_1 X_2 \ldots X_n$ (a string of terminals and nonterminals) as follows:

1. $\text{First}(\alpha)$ contains $\text{First}(X_1) - \{\varepsilon\}$.
2. For each $i = 2, \ldots, n$, if $\text{First}(X_k)$ contains $\varepsilon$ for all $k = 1, \ldots, i - 1$, then $\text{First}(\alpha)$ contains $\text{First}(X_i) - \{\varepsilon\}$.
3. Finally, if for all $i = 1, \ldots, n$, $\text{First}(X_i)$ contains $\varepsilon$, then $\text{First}(\alpha)$ contains $\varepsilon$. 
Algorithm for computing First(A) for nonterminals A

for all nonterminals A do First(A):=\{\};
while there are changes to any First(A) do
    for each production choice A \rightarrow X_1...X_n do
        k := 1; Continue:=true;
        while Continue = true and k <= n do
            add First(X_k)−\{ε\} to First(A);
            if ε not in First(X_k) then Continue:=false;
            k := k + 1;
        if Continue = true then add ε to First(A);
Algorithm for computing First(A) for nonterminals A

for all nonterminals A do
  First(A):=\{
while there are changes to any First(A) do
  for each production choice A \rightarrow X_1...X_n do
    k := 1; Continue:=true;
    while Continue = true and k <= n do
      add First(X_k)−\{\varepsilon\} to First(A);
      if \varepsilon\ not in First(X_k) then Continue:=false;
      k := k + 1;
    if Continue = true then add \varepsilon\ to First(A);

Simplified algorithm for First Sets in the absence of \varepsilon\-productions:

LL(1) parsing
Algorithm for computing First(A) for nonterminals A

for all nonterminals A do
  First(A):=\{\};
while there are changes to any First(A) do
  for each production choice A → X₁…Xₙ do
    k := 1; Continue:=true;
    while Continue = true and k <= n do
      add First(Xₖ)−{ε} to First(A);
      if ε not in First(Xₖ) then Continue:=false;
      k := k + 1;
    if Continue = true then add ε to First(A);

Simplified algorithm for First Sets in the absence of ε-productions:
for all nonterminals A do
  First(A):=\{\};
while there are changes to any First(A) do
  for each production choice A → X₁…Xₙ do
    add First(X₁) to First(A);

LL(1) parsing
Consider the simple integer expression grammar:

\[
\begin{align*}
\text{exp} & \rightarrow \text{exp addop term} \mid \text{term} \\
\text{addop} & \rightarrow + \mid - \\
\text{term} & \rightarrow \text{term mulop factor} \mid \text{factor} \\
\text{mulop} & \rightarrow * \\
\text{factor} & \rightarrow ( \text{exp} ) \mid \text{number}
\end{align*}
\]
Consider the simple integer expression grammar:

\[
\begin{align*}
\text{exp} & \rightarrow \text{exp addop term} \mid \text{term} \\
\text{addop} & \rightarrow + \mid - \\
\text{term} & \rightarrow \text{term mulop factor} \mid \text{factor} \\
\text{mulop} & \rightarrow * \\
\text{factor} & \rightarrow ( \text{exp} ) \mid \text{number}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Grammar rule</th>
<th>Pass 1</th>
<th>Pass 2</th>
<th>Pass 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{exp} \rightarrow \text{exp addop term}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\text{exp} \rightarrow \text{term}</td>
<td></td>
<td></td>
<td>\text{First}(\text{exp}) = { (, \text{number} }</td>
</tr>
<tr>
<td>\text{addop} \rightarrow +</td>
<td></td>
<td>\text{First}(\text{addop}) = { + }</td>
<td></td>
</tr>
<tr>
<td>\text{addop} \rightarrow -</td>
<td></td>
<td>\text{First}(\text{addop}) = { +, - }</td>
<td></td>
</tr>
<tr>
<td>\text{term} \rightarrow \text{term mulop factor}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\text{term} \rightarrow \text{factor}</td>
<td></td>
<td>\text{First}(\text{term}) = { (, \text{number} }</td>
<td></td>
</tr>
<tr>
<td>\text{mulop} \rightarrow *</td>
<td></td>
<td>\text{First}(\text{mulop}) = { * }</td>
<td></td>
</tr>
<tr>
<td>\text{factor} \rightarrow ( \text{exp} )</td>
<td></td>
<td>\text{First}(\text{factor}) = { ( }</td>
<td></td>
</tr>
<tr>
<td>\text{factor} \rightarrow \text{number}</td>
<td></td>
<td>\text{First}(\text{factor}) = { (, \text{number} }</td>
<td></td>
</tr>
</tbody>
</table>
Thus:
First($exp$) = {($, number$}
First($term$) = {($, number$}
First($factor$) = {($, number$}
First($addop$) = {+, −}
First($mulop$) = {*}
Consider the grammar:

\[
\begin{align*}
    &\text{statement} \rightarrow \text{if-stmt} \mid \text{other} \\
    &\text{if-stmt} \rightarrow \text{if} (\exp) \text{statement} \text{else-part} \\
    &\text{else-part} \rightarrow \text{else} \text{statement} \mid \varepsilon \\
    &\exp \rightarrow 0 \mid 1
\end{align*}
\]
Consider the grammar:

\[
\begin{align*}
\text{statement} & \rightarrow \text{if-stmt} \mid \text{other} \\
\text{if-stmt} & \rightarrow \text{if} \ (\ \text{exp} \ ) \ \text{statement} \ \text{else-part} \\
\text{else-part} & \rightarrow \text{else} \ \text{statement} \mid \varepsilon \\
\text{exp} & \rightarrow 0 \mid 1
\end{align*}
\]

<table>
<thead>
<tr>
<th>Grammar rule</th>
<th>Pass 1</th>
<th>Pass 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{statement} \rightarrow \text{if-stmt}</td>
<td></td>
<td>First(\text{statement}) = {\text{if, other}}</td>
</tr>
<tr>
<td>\text{statement} \rightarrow \text{other}</td>
<td>First(\text{statement}) = {\text{other}}</td>
<td></td>
</tr>
<tr>
<td>\text{if-stmt} \rightarrow \text{if} \ (\ \text{exp} \ ) \ \text{statement} \ \text{else-part}</td>
<td>First(\text{if-stmt}) = {\text{if}}</td>
<td></td>
</tr>
<tr>
<td>\text{else-part} \rightarrow \text{else} \ \text{statement}</td>
<td>First(\text{else-part}) = {\text{else}}</td>
<td></td>
</tr>
<tr>
<td>\text{else-part} \rightarrow \varepsilon</td>
<td>First(\text{else-part}) = {\text{else, }\varepsilon}</td>
<td></td>
</tr>
<tr>
<td>\text{exp} \rightarrow 0</td>
<td>First(\text{exp}) = {0}</td>
<td></td>
</tr>
<tr>
<td>\text{exp} \rightarrow 1</td>
<td>First(\text{exp}) = {0, 1}</td>
<td></td>
</tr>
</tbody>
</table>

LL(1) parsing
Thus:
First(\textit{statement}) = \{\textit{if}, \textit{other}\}
First(\textit{if-stmt}) = \{\textit{if}\}
First(\textit{else-part}) = \{\textit{else}, \varepsilon\}
First(\textit{exp}) = \{0, 1\}
Follow sets - definition

Given a nonterminal $A$, the follow set $\text{Follow}(A)$, consisting of terminals, and possibly $\$, is defined as follows:

1. If $A$ is the start symbol, then $\$ is in $\text{Follow}(A)$.
2. If there is a production $B \rightarrow \alpha A \gamma$, then $\text{First}(\gamma) - \{\varepsilon\}$ is in $\text{Follow}(A)$.
3. If there is a production $B \rightarrow \alpha A \gamma$ such that $\varepsilon$ is in $\text{First}(\gamma)$, then $\text{Follow}(A)$ contains $\text{Follow}(B)$. 

LL(1) parsing
Given a nonterminal $A$, the follow set $\text{Follow}(A)$, consisting of terminals, and possibly $\$$, is defined as follows:

1. If $A$ is the start symbol, then $\$$ is in $\text{Follow}(A)$.
Follow sets - definition

Given a nonterminal $A$, the follow set $\text{Follow}(A)$, consisting of terminals, and possibly $\$$, is defined as follows:

1. If $A$ is the start symbol, then $\$$ is in $\text{Follow}(A)$.
2. If there is a production $B \to \alpha A \gamma$, then $\text{First}(\gamma) - \{\varepsilon\}$ is in $\text{Follow}(A)$.
3. If there is a production $B \to \alpha A \gamma$ such that $\varepsilon$ is in $\text{First}(\gamma)$, then $\text{Follow}(A)$ contains $\text{Follow}(B)$.
Follow sets - definition

Given a nonterminal $A$, the follow set $\text{Follow}(A)$, consisting of terminals, and possibly $\$, is defined as follows:

1. If $A$ is the start symbol, then $\$ is in $\text{Follow}(A)$.
2. If there is a production $B \rightarrow \alpha A \gamma$, then $\text{First}(\gamma) \setminus \{\varepsilon\}$ is in $\text{Follow}(A)$.
3. If there is a production $B \rightarrow \alpha A \gamma$ such that $\varepsilon$ is in $\text{First}(\gamma)$, then $\text{Follow}(A)$ contains $\text{Follow}(B)$.
Follow(start-symbol):= \{\$\};
for all nonterminals \( A \neq \) start-symbol do Follow(A):=\{\};
while there are changes to any Follow sets do
    for each production \( A \rightarrow X_1...X_n \) do
        for each \( X_i \) that is a nonterminal do
            add First(\( X_{i+1}...X_n \))\(-\{\varepsilon\}\) to Follow(\( X_i \))
            (* Note: if \( i = n \), then \( X_{i+1}...X_n = \varepsilon \) *)
            if \( \varepsilon \) is in First(\( X_{i+1}...X_n \)) then
                add Follow(A) to Follow(\( X_i \))
Follow sets example 1

We consider again the grammar:

\[\begin{align*}
  \text{exp} & \rightarrow \text{exp addop term} \mid \text{term} \\
  \text{addop} & \rightarrow + \mid - \\
  \text{term} & \rightarrow \text{term mulop factor} \mid \text{factor} \\
  \text{mulop} & \rightarrow * \\
  \text{factor} & \rightarrow ( \text{exp} ) \mid \text{number}
\end{align*}\]
We consider again the grammar:

\[
\begin{align*}
exp & \rightarrow \text{exp addop term} \mid \text{term} \\
addop & \rightarrow + \mid - \\
term & \rightarrow \text{term mulop factor} \mid \text{factor} \\
mulop & \rightarrow * \\
factor & \rightarrow ( \text{exp} ) \mid \text{number}
\end{align*}
\]

Recall that:

\[
\begin{align*}
\text{First}(\text{exp}) &= \{ (, \text{number} \} \\
\text{First}(\text{term}) &= \{ (, \text{number} \} \\
\text{First}(\text{factor}) &= \{ (, \text{number} \} \\
\text{First}(\text{addop}) &= \{ +, - \} \\
\text{First}(\text{mulop}) &= \{ * \}
\end{align*}
\]
In the computation of the Follow sets for the grammar we omit the four grammar rule choices that have no possibility of affecting the computation.
In the computation of the Follow sets for the grammar we omit the four grammar rule choices that have no possibility of affecting the computation.

<table>
<thead>
<tr>
<th>Grammar rule</th>
<th>Pass 1</th>
<th>Pass 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp → exp addop term</td>
<td>Follow(exp) =</td>
<td>Follow(term) =</td>
</tr>
<tr>
<td></td>
<td>{$, +, −}</td>
<td>{$, +, −, *, }</td>
</tr>
<tr>
<td></td>
<td>Follow(addop) =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{ (, number}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Follow(term) =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{$, +, − }</td>
<td></td>
</tr>
<tr>
<td>exp → term</td>
<td>Follow(term) =</td>
<td>Follow(factor) =</td>
</tr>
<tr>
<td></td>
<td>{$, +, −, * }</td>
<td>{$, +, −, *, }</td>
</tr>
<tr>
<td>term → term mulop factor</td>
<td>Follow(term) =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{$, +, −, * }</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Follow(mulop) =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{ (, number}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Follow(factor) =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{$, +, −, * }</td>
<td></td>
</tr>
<tr>
<td>term → factor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>factor → ( exp )</td>
<td>Follow(exp) =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>= { $, +, −, }</td>
<td></td>
</tr>
</tbody>
</table>
Thus:
\[
\text{Follow}(\text{exp}) = \{\$, +, -, \}\}
\text{Follow}(\text{term}) = \{\$, +, -, *, \}\}
\text{Follow}(\text{factor}) = \{\$, +, -, *, \}\}
\text{Follow}(\text{addop}) = \{(, \text{number}\}
\text{Follow}(\text{mulop}) = \{(, \text{number}\}
\]
LL(1) parsing example

\[
\begin{align*}
\textit{statement} & \rightarrow \textit{if-stmt} \mid \textit{other} \\
\textit{if-stmt} & \rightarrow \textbf{if} \ ( \textit{exp} \ ) \ \textit{statement} \ \textit{else-part} \\
\textit{else-part} & \rightarrow \textbf{else} \ \textit{statement} \mid \epsilon \\
\textit{exp} & \rightarrow 0 \mid 1
\end{align*}
\]
LL(1) parsing example

- \( \text{statement} \rightarrow \text{if-stmt} \mid \text{other} \)
  - \( \text{if-stmt} \rightarrow \text{if} ( \text{exp} ) \text{ statement else-part} \)
  - \( \text{else-part} \rightarrow \text{else statement} \mid \epsilon \)
  - \( \text{exp} \rightarrow 0 \mid 1 \)

- Recall that:
  - First(\( \text{statement} \)) = \{\text{if, other}\}
  - First(\( \text{if-stmt} \)) = \{\text{if}\}
  - First(\( \text{else-part} \)) = \{\text{else, } \epsilon\}
  - First(\( \text{exp} \)) = \{0, 1\}
LL(1) parsing example

\[
\text{statement} \rightarrow \text{if-stmt} \mid \text{other}
\]
\[
\text{if-stmt} \rightarrow \text{if ( exp ) statement else-part}
\]
\[
\text{else-part} \rightarrow \text{else statement} \mid \varepsilon
\]
\[
\text{exp} \rightarrow 0 \mid 1
\]

▶ Recall that:
\[
\text{First}(\text{statement}) = \{\text{if, other}\}
\]
\[
\text{First}(\text{if-stmt}) = \{\text{if}\}
\]
\[
\text{First}(\text{else-part}) = \{\text{else, } \varepsilon\}
\]
\[
\text{First}(\text{exp}) = \{0, 1\}
\]

▶ One can verify that:
\[
\text{Follow}(\text{statement}) = \{\$, \text{else}\}
\]
\[
\text{Follow}(\text{if-stmt}) = \{\$, \text{else}\}
\]
\[
\text{Follow}(\text{else-part}) = \{\$, \text{else}\}
\]
\[
\text{Follow}(\text{exp}) = \{ \) \}
\]
Recall that the LL(1) parsing table \( M[N, T] \) is constructed by repeating the following two steps for each nonterminal \( A \) and each production \( A \rightarrow \alpha \):

1. For each token \( a \) in First(\( \alpha \)), add \( A \rightarrow \alpha \) to the entry \( M[A, a] \).
2. If \( \varepsilon \) is in First(\( \alpha \)), for each element \( a \) of Follow(\( A \)) (where \( a \) is a token or \( a \) is $), add \( A \rightarrow \alpha \) to \( M[A, a] \).
Recall that the LL(1) parsing table $M[N, T]$ is constructed by repeating the following two steps for each nonterminal $A$ and each production $A \to \alpha$:

1. For each token $a$ in $\text{First}(\alpha)$, add $A \to \alpha$ to the entry $M[A, a]$. 
Recall that the LL(1) parsing table $M[N, T]$ is constructed by repeating the following two steps for each nonterminal $A$ and each production $A \rightarrow \alpha$:

1. For each token $a$ in $\text{First}(\alpha)$, add $A \rightarrow \alpha$ to the entry $M[A, a]$.
2. If $\varepsilon$ is in $\text{First}(\alpha)$, for each element $a$ of $\text{Follow}(A)$ (where $a$ is a token or $a$ is $\$$), add $A \rightarrow \alpha$ to $M[A, a]$. 
Using the procedure on the previous slide, we obtain the following table:
Using the procedure on the previous slide, we obtain the following table:

<table>
<thead>
<tr>
<th>$M[N, T]$</th>
<th>if</th>
<th>other</th>
<th>else</th>
<th>0</th>
<th>1</th>
<th>$$$</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>statement</code></td>
<td><code>if-stmt</code></td>
<td><code>statement</code></td>
<td><code>else</code></td>
<td>0</td>
<td>1</td>
<td>$$$</td>
</tr>
<tr>
<td><code>if-stmt</code></td>
<td><code>if-stmt</code></td>
<td><code>else-part</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>else-part</code></td>
<td><code>else-part</code></td>
<td><code>else-part</code></td>
<td><code>exp</code></td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><code>exp</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- $\rightarrow$ denotes production rules.
We notice, that as expected, this grammar is not LL(1), since the entry $M[else-part,else]$ contains two entries, corresponding to the dangling else ambiguity.
We notice, that as expected, this grammar is not LL(1), since the entry $M[else-part, else]$ contains two entries, corresponding to the dangling else ambiguity.

We could apply the disambiguating rule that would always prefer the rule that generates the current lookahead token over any other (this corresponds to the most closely nested disambiguating rule), and thus the production

$else-part \rightarrow else \ statement$
We notice, that as expected, this grammar is not LL(1), since the entry \( M[\text{else-part, else}] \) contains two entries, corresponding to the dangling else ambiguity.

We could apply the disambiguating rule that would always prefer the rule that generates the current lookahead token over any other (this corresponds to the most closely nested disambiguating rule), and thus the production

\[
\text{else-part} \rightarrow \text{else statement}
\]

We now show the LL(1) parsing actions for the string

\[
\text{if (0) if(1) other else other}
\]
We notice, that as expected, this grammar is not LL(1), since the entry $M[else-part, else]$ contains two entries, corresponding to the dangling else ambiguity.

We could apply the disambiguating rule that would always prefer the rule that generates the current lookahead token over any other (this corresponds to the most closely nested disambiguating rule), and thus the production

$$else-part \rightarrow else \; statement$$

We now show the LL(1) parsing actions for the string

```
if (0) if(1) other else other
```

We use the following abbreviations:

- $statement = S$
- $if-stmt = I$
- $else-part = L$
- $exp = E$
- $if = i$
- $else = e$
- $other = o$
LL(1) parsing example continue

<table>
<thead>
<tr>
<th>Parsing stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>i (0) i (1) o e o$</td>
<td>$S \to I$</td>
</tr>
<tr>
<td>$I$</td>
<td>i (0) i (1) o e o$</td>
<td>$I \to i ( E ) S L$</td>
</tr>
<tr>
<td>$LS)E(i$</td>
<td>i (0) i (1) o e o$</td>
<td>match</td>
</tr>
<tr>
<td>$LS)E(0$</td>
<td>(0) i (1) o e o$</td>
<td>match</td>
</tr>
<tr>
<td>$LS)E$</td>
<td>0) i (1) o e o$</td>
<td>$E \to 0$</td>
</tr>
<tr>
<td>$LS)0$</td>
<td>0) i (1) o e o$</td>
<td>match</td>
</tr>
<tr>
<td>$LS)$</td>
<td>) i (1) o e o$</td>
<td>match</td>
</tr>
<tr>
<td>$LS$</td>
<td>i (1) o e o$</td>
<td>$S \to I$</td>
</tr>
<tr>
<td>$LI$</td>
<td>i (1) o e o$</td>
<td>$I \to i ( E ) S L$</td>
</tr>
<tr>
<td>$LLS)E(i$</td>
<td>i (1) o e o$</td>
<td>match</td>
</tr>
<tr>
<td>$LLS)E(1$</td>
<td>(1) o e o$</td>
<td>match</td>
</tr>
<tr>
<td>$LLS)E$</td>
<td>1) o e o$</td>
<td>$E \to 1$</td>
</tr>
<tr>
<td>$LLS)1$</td>
<td>1) o e o$</td>
<td>match</td>
</tr>
<tr>
<td>$LLS)$</td>
<td>) o e o$</td>
<td>match</td>
</tr>
<tr>
<td>$LLS$</td>
<td>o e o$</td>
<td>$S \to o$</td>
</tr>
<tr>
<td>$LLo$</td>
<td>o e o$</td>
<td>match</td>
</tr>
<tr>
<td>$LL$</td>
<td>e o$</td>
<td>$L \to e S$</td>
</tr>
<tr>
<td>$LS$e$</td>
<td>e o$</td>
<td>match</td>
</tr>
<tr>
<td>$LS$</td>
<td>o$</td>
<td>$S \to o$</td>
</tr>
<tr>
<td>$Lo$</td>
<td>o$</td>
<td>match</td>
</tr>
<tr>
<td>$L$</td>
<td>$</td>
<td>$L \to \varepsilon$</td>
</tr>
<tr>
<td>$</td>
<td>$</td>
<td>accept</td>
</tr>
</tbody>
</table>
Finally we consider the LL(1) parsing example discussed in the JFLAP tutorial at http://www.jflap.org/tutorial/

- S → a A B b
- A → a A c
- A → λ
- B → b B
- B → c

Now we use JFLAP to parse aacbbcb
Finally we consider the LL(1) parsing example discussed in the JFLAP tutorial at http://www.jflap.org/tutorial/

We use JFLAP to calculate First and Follow sets and the LL(1) parse table for the grammar:

\[
S \rightarrow a \ A \ B \ b \\
A \rightarrow a \ A \ c \\
A \rightarrow \lambda \\
B \rightarrow b \ B \\
B \rightarrow c
\]
Finally we consider the LL(1) parsing example discussed in the JFLAP tutorial at http://www.jflap.org/tutorial/

We use JFLAP to calculate First and Follow sets and the LL(1) parse table for the grammar:

\[ S \rightarrow a\; A\; B\; b \]
\[ A \rightarrow a\; A\; c \]
\[ A \rightarrow \lambda \]
\[ B \rightarrow b\; B \]
\[ B \rightarrow c \]

Now we use JFLAP to parse \( aacbbcb \)