CS 314 Principles of Programming Languages

Lecture 8: LL(1) Parsing

Zheng (Eddy) Zhang

Rutgers University

February 12, 2018
Homework 3 posted, due Sunday 2/18 11:55 pm EST.
Key Property:

Whenever two productions $A ::= \alpha$ and $A ::= \beta$ both appear in the grammar, we would like

\[ \text{FIRST} (\alpha) \cap \text{FIRST} (\beta) = \emptyset, \text{ and} \]
\[ \text{if } \alpha \Rightarrow^* \varepsilon, \text{ then } \text{FIRST} (\beta) \cap \text{FOLLOW} (A) = \emptyset \]

Analogue case for $\beta \Rightarrow^* \varepsilon$.

Note: due to first condition, at most one of $\alpha$ and $\beta$ can derive $\varepsilon$.

This would allow the parser to make a correct choice with a lookahead of only one symbol!
FIRST and FOLLOW Sets

FIRST(α):

For some $\alpha \in (T \cup NT \cup EOF \cup \varepsilon)^*$, define FIRST(α) as the set of tokens that appear as the first symbol in some string that derives from $\alpha$.

That is, $x \in \text{FIRST}(\alpha)$ iff $\alpha \Rightarrow^* x\gamma$ for some $\gamma$

FIRST set is defined over the strings of grammar symbols $(T \cup NT \cup EOF \cup \varepsilon)^*$

T: terminals  NT: non-terminals
First Set Example

Start ::= S $\texttt{eof}$
S ::= a S b | $\varepsilon$

\[ FIRST(aSb) = \{a\} \]
\[ FIRST(\varepsilon) = \{\varepsilon\} \]
\[ FIRST(S) = \{a, \varepsilon\} \]
FOLLOW(A):

For $A \in NT$, define $\text{FOLLOW}(A)$ as the set of tokens that can occur immediately after $A$ in a valid sentential form.

$\text{FOLLOW}$ set is defined over the set of non-terminal symbols, $NT$. 
Start ::= S eof
S ::= a S b | ε

\( FOLLOW(S) = \{ \text{eof}, b \} \)
Build FIRST(X) for all grammar symbols X:

- For each X as a terminal, then FIRST(X) is \{X\}
- If X ::= ε, then ε ∈ FIRST(X)
- For each X as a non-terminal, initialize FIRST(X) to ∅
- **Iterate until** no more terminals or ε can be added to any FIRST(X):
  - For each rule in the grammar of the form X ::= Y₁ Y₂…Yₖ
    - add a to FIRST(X) if a ∈ FIRST(Y₁)
    - add a to FIRST(X) if a ∈ FIRST(Yᵢ) and ε ∈ FIRST(Yⱼ)
      for all 1 ≤ j ≤ i-1 and i ≥ 2
    - add ε to FIRST(X) if ε ∈ FIRST(Yᵢ) for all 1 ≤ i ≤ k

EndFor

**End iterate**
Filling in the Details: Computing \textit{FIRST} sets

for each $x \in (T \cup \text{EOF} \cup \varepsilon)$
\begin{align*}
\textit{FIRST}(x) & \leftarrow \{x\} \\
\end{align*}
for each $A \in \text{NT}$, $\textit{FIRST}(A) \leftarrow \emptyset$

Initially, set \textit{FIRST} for each terminal symbol, EOF and $\varepsilon$

while (\textit{FIRST} sets are still changing) do
    for each $p \in P$, of the form $X \rightarrow Y_1 Y_2 \ldots Y_k$ do
        temp $\leftarrow \textit{FIRST}(Y_1) - \{\varepsilon\}$
        i $\leftarrow$ 1
        while ($i \leq k$-1 and $\varepsilon \in \textit{FIRST}(Y_i)$)
            temp $\leftarrow$ temp $\cup$ ($\textit{FIRST}(Y_{i+1}) - \{\varepsilon\}$)
            i $\leftarrow$ i + 1
        end // while loop
        if $i == k$ and $\varepsilon \in \textit{FIRST}(Y_k)$
            then temp $\leftarrow$ temp $\cup$ \{\varepsilon\}
            $\textit{FIRST}(X) \leftarrow \textit{FIRST}(X) \cup$ temp
        end // if - then
    end // for loop
end // while loop
Filling in the Details: Computing $FIRST$ sets

for each $x \in (T \cup EOF \cup \varepsilon)$

$FIRST(x) \leftarrow \{x\}$

for each $A \in NT$, $FIRST(A) \leftarrow \emptyset$

while ($FIRST$ sets are still changing) do

for each $p \in P$, of the form $X \rightarrow Y_1 Y_2 \ldots Y_k$ do

$\text{temp} \leftarrow FIRST(Y_1) - \{\varepsilon\}$

$i \leftarrow 1$

while ($i \leq k-1$ and $\varepsilon \in FIRST(Y_i)$)

$\text{temp} \leftarrow \text{temp} \cup (FIRST(Y_{i+1}) - \{\varepsilon\})$

$i \leftarrow i + 1$

end // while loop

if $i == k$ and $\varepsilon \in FIRST(Y_k)$

then $\text{temp} \leftarrow \text{temp} \cup \{\varepsilon\}$

$FIRST(X) \leftarrow FIRST(X) \cup \text{temp}$

end // if - then

end // for loop

end // while loop

$\varepsilon$ complicates matters

If $FIRST(Y_1)$ contains $\varepsilon$, then we need to add $FIRST(Y_2)$ to rhs, and …

If the entire rhs can go to $\varepsilon$, then we add $\varepsilon$ to $FIRST$(lhs)
Computing \textit{FIRST} sets

for each \( x \in (T \cup \text{EOF} \cup \epsilon) \)
\[
\text{\textit{FIRST}}(x) \leftarrow \{x\}
\]
for each \( A \in \text{NT} \), \( \text{\textit{FIRST}}(A) \leftarrow \emptyset \)

while \( (\text{\textit{FIRST}} \text{ sets are still changing}) \) do
\begin{enumerate}
\item for each \( p \in P \), of the form \( X \rightarrow Y_1 Y_2 \ldots Y_k \) do
\item temp \( \leftarrow \text{\textit{FIRST}}(Y_1) - \{\epsilon\} \)
\item i \( \leftarrow 1 \)
\item while \( (i \leq k - 1 \text{ and } \epsilon \in \text{\textit{FIRST}}(Y_i)) \) do
\item temp \( \leftarrow \text{temp} \cup (\text{\textit{FIRST}}(Y_{i+1}) - \{\epsilon\}) \)
\item i \( \leftarrow i + 1 \)
\item end // while loop
\item if i == k and \( \epsilon \in \text{\textit{FIRST}}(Y_k) \)
\item then temp \( \leftarrow \text{temp} \cup \{\epsilon\} \)
\item \( \text{\textit{FIRST}}(X) \leftarrow \text{\textit{FIRST}}(X) \cup \text{temp} \)
\item end // if - then
\item end // for loop
\item end // while loop
\end{enumerate}

Outer loop is monotone increasing for \textit{FIRST} sets
\[ \Rightarrow |T \cup \text{NT} \cup \text{EOF} \cup \epsilon| \text{ is bounded, so it terminates} \]
Example

Consider the SheepNoise grammar and its \textit{FIRST} sets

\[
\begin{align*}
\text{Goal} & ::= \text{SheepNoise} \\
\text{SheepNoise} & ::= \text{SheepNoise} \text{ baa} | \text{ baa}
\end{align*}
\]

Clearly, \(\text{FIRST}(x) = \{\text{baa}\}, \forall x \in (T \cup NT)\)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>\textit{FIRST} Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>baa</td>
</tr>
<tr>
<td>SheepNoise</td>
<td>baa</td>
</tr>
<tr>
<td>baa</td>
<td>baa</td>
</tr>
</tbody>
</table>

\text{baa is a terminal symbol}
Computing $FIRST$ sets

for each $x \in (T \cup \text{EOF} \cup \epsilon)$

$FIRST(x) \leftarrow \{x\}$

for each $A \in \text{NT}$, $FIRST(A) \leftarrow \emptyset$

Initialization assigns each $FIRST$ set a value

while ($FIRST$ sets are still changing) do

for each $p \in P$, of the form $X \rightarrow Y_1 Y_2 \ldots Y_k$ do

$\text{temp} \leftarrow FIRST(Y_1) - \{\epsilon\}$

$i \leftarrow 1$

while ($i \leq k-1$ and $\epsilon \in FIRST(Y_i)$)

$\text{temp} \leftarrow \text{temp} \cup (FIRST(Y_{i+1}) - \{\epsilon\})$

$i \leftarrow i + 1$

end // while loop

if $i == k$ and $\epsilon \in FIRST(Y_k)$

then $\text{temp} \leftarrow \text{temp} \cup \{\epsilon\}$

$FIRST(X) \leftarrow FIRST(X) \cup \text{temp}$

end // if - then

end // for loop

end // while loop

Symbol | $FIRST$ Set
---|---
Goal |  
SheepNoise |  
baa | 
Computing \textit{FIRST} sets

for each $x \in (T \cup \text{EOF} \cup \varepsilon)$
\[
\text{\textit{FIRST}}(x) \leftarrow \{x\}
\]
for each $A \in \text{NT}$, \textit{FIRST}(A) $\leftarrow \emptyset$

while (\textit{FIRST} sets are still changing) do
\begin{itemize}
  \item for each $p \in P$, of the form $X \rightarrow Y_1 Y_2 \ldots Y_k$ do
    \begin{itemize}
      \item \textit{temp} $\leftarrow$ \textit{FIRST}(Y_1) - \{\varepsilon\}
      \item \textit{i} $\leftarrow$ 1
      \item while (\textit{i} $\leq$ \textit{k}-1 and \textit{\varepsilon} $\in$ \textit{FIRST}(Y_i))
        \begin{itemize}
          \item \textit{temp} $\leftarrow$ \textit{temp} $\cup$ (\textit{FIRST}(Y_{i+1}) - \{\varepsilon\})
          \item \textit{i} $\leftarrow$ \textit{i} + 1
        \end{itemize}
      \item end // while loop
      \item \text{\textit{i}} $\leftarrow$ \textit{k} and \textit{\varepsilon} $\in$ \textit{FIRST}(Y_k)
      \item then \textit{temp} $\leftarrow$ \textit{temp} $\cup$ \{\varepsilon\}
      \item \text{\textit{FIRST}}(X) $\leftarrow$ \textit{FIRST}(X) $\cup$ \textit{temp}
    \end{itemize}
end // for loop
\end{itemize}
end // while loop

\begin{tabular}{|c|c|}
\hline
Symbol & \textit{FIRST} Set \\
\hline
Goal & $\emptyset$ \\
SheepNoise & $\{\text{baa}\}$ \\
\texttt{baa} & $\{\text{baa}\}$ \\
\hline
\end{tabular}

If we visit the rule in the order 3, 2, 1
Computing $FIRST$ sets

for each $x \in (\text{T} \cup \text{EOF} \cup \varepsilon)$

$FIRST(x) \leftarrow \{x\}$

for each $A \in \text{NT}$, $FIRST(A) \leftarrow \emptyset$

while ($FIRST$ sets are still changing) do

for each $p \in P$, of the form $X \rightarrow Y_1Y_2\ldots Y_k$ do

$\text{temp} \leftarrow FIRST(Y_1) - \{\varepsilon\}$

$i \leftarrow 1$

while ($i \leq k-1$ and $\varepsilon \in FIRST(Y_i)$)

$\text{temp} \leftarrow \text{temp} \cup (FIRST(Y_{i+1}) - \{\varepsilon\}))$

$i \leftarrow i + 1$

end // while loop

if $i == k$ and $\varepsilon \in FIRST(Y_k)$

then $\text{temp} \leftarrow \text{temp} \cup \{\varepsilon\}$

$FIRST(X) \leftarrow FIRST(X) \cup \text{temp}$

end // if - then

end // for loop

end // while loop

1 Goal ::= SheepNoise
2 SheepNoise ::= SheepNoise baa
3 SheepNoise ::= baa

If we visit the rule in the order 3, 2, 1

<table>
<thead>
<tr>
<th>Symbol</th>
<th>$FIRST$ Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td></td>
</tr>
<tr>
<td>SheepNoise</td>
<td>{baa}</td>
</tr>
<tr>
<td>baa</td>
<td>{baa}</td>
</tr>
</tbody>
</table>
An Example

Consider the simplest parentheses grammar

1. Goal ::= List
2. List ::= Pair List
3. | ε
4. Pair ::= LP List RP

Where LP is ( and RP is )

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Initial</th>
<th>1st</th>
<th>2nd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>∅</td>
<td></td>
<td></td>
</tr>
<tr>
<td>List</td>
<td>∅</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pair</td>
<td>∅</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LP</td>
<td>LP</td>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>RP</td>
<td>RP</td>
<td>RP</td>
<td>RP</td>
</tr>
<tr>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
</tr>
</tbody>
</table>
An Example

Consider the simplest parentheses grammar

<table>
<thead>
<tr>
<th>Rule</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Goal ::= List</td>
</tr>
<tr>
<td>2</td>
<td>List ::= Pair List</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Pair ::= LP List RP</td>
</tr>
</tbody>
</table>

Where LP is ( and RP is )

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Initial</th>
<th>1&lt;sup&gt;st&lt;/sup&gt;</th>
<th>2&lt;sup&gt;nd&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>Ø</td>
<td></td>
<td></td>
</tr>
<tr>
<td>List</td>
<td>Ø</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pair</td>
<td>Ø</td>
<td>LP</td>
<td></td>
</tr>
<tr>
<td>LP</td>
<td>LP</td>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>RP</td>
<td>RP</td>
<td>RP</td>
<td>RP</td>
</tr>
<tr>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
</tr>
</tbody>
</table>

If we visit the rules in order 4, 3, 2, 1 ⇒
An Example

Consider the simplest parentheses grammar

1. Goal ::= List
2. List ::= Pair List | ε
3. Pair ::= LP List RP

Where LP is ( and RP is )

If we visit the rules in order 4, 3, 2, 1 ⇒

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Initial</th>
<th>1st</th>
<th>2nd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>Ø</td>
<td></td>
<td></td>
</tr>
<tr>
<td>List</td>
<td>Ø</td>
<td>LP, ε</td>
<td></td>
</tr>
<tr>
<td>Pair</td>
<td>Ø</td>
<td>LP</td>
<td></td>
</tr>
<tr>
<td>LP</td>
<td>LP</td>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>RP</td>
<td>RP</td>
<td>RP</td>
<td>RP</td>
</tr>
<tr>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
</tr>
</tbody>
</table>
Consider the simplest parentheses grammar

\[
\text{Goal ::= List} \\
\text{List ::= Pair List} \\
\quad \mid \varepsilon \\
\text{Pair ::= LP List RP}
\]

Where LP is ( and RP is )

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Initial</th>
<th>1&lt;sup&gt;st&lt;/sup&gt;</th>
<th>2&lt;sup&gt;nd&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>∅</td>
<td>LP, ε</td>
<td></td>
</tr>
<tr>
<td>List</td>
<td>∅</td>
<td>LP, ε</td>
<td></td>
</tr>
<tr>
<td>Pair</td>
<td>∅</td>
<td>LP</td>
<td></td>
</tr>
<tr>
<td>LP</td>
<td>LP</td>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>RP</td>
<td>RP</td>
<td>RP</td>
<td>RP</td>
</tr>
<tr>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
</tr>
</tbody>
</table>

If we visit the rules in order 4, 3, 2, 1 ➔
Consider the simplest parentheses grammar

1. Goal ::= List
2. List ::= Pair List
3. | ε
4. Pair ::= LP List RP

Where LP is ( and RP is )

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Initial</th>
<th>1st</th>
<th>2nd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>⊘</td>
<td>LP, ε</td>
<td></td>
</tr>
<tr>
<td>List</td>
<td>⊘</td>
<td>LP, ε</td>
<td></td>
</tr>
<tr>
<td>Pair</td>
<td>⊘</td>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>LP</td>
<td>LP</td>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>RP</td>
<td>RP</td>
<td>RP</td>
<td>RP</td>
</tr>
<tr>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
</tr>
</tbody>
</table>

If we visit the rules in order 4, 3, 2, 1  ⇒
An Example

Consider the simplest parentheses grammar

1. \text{Goal} ::= \text{List}  \\
2. \text{List} ::= \text{Pair List}  \\
3. \phantom{1} | \phantom{1} \varepsilon  \\
4. \text{Pair} ::= \text{LP List RP}  \\

Where \text{LP} is ( and \text{RP} is )

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Initial</th>
<th>1\text{st}</th>
<th>2\text{nd}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>\varnothing</td>
<td>LP, \varepsilon</td>
<td></td>
</tr>
<tr>
<td>List</td>
<td>\varnothing</td>
<td>LP, \varepsilon</td>
<td>LP, \varepsilon</td>
</tr>
<tr>
<td>Pair</td>
<td>\varnothing</td>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>LP</td>
<td>LP</td>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>RP</td>
<td>RP</td>
<td>RP</td>
<td>RP</td>
</tr>
<tr>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
</tr>
</tbody>
</table>

If we visit the rules in order 4, 3, 2, 1 \implies
Consider the simplest parentheses grammar

1. **Goal ::= List**
2. **List ::= Pair List**
3. | ε
4. **Pair ::= LP List RP**

Where **LP** is ( and **RP** is )

If we visit the rules in order 4, 3, 2, 1 \[\Rightarrow\]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Initial</th>
<th>1st</th>
<th>2nd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>∅</td>
<td>LP, ε</td>
<td>LP, ε</td>
</tr>
<tr>
<td>List</td>
<td>∅</td>
<td>LP, ε</td>
<td>LP, ε</td>
</tr>
<tr>
<td>Pair</td>
<td>∅</td>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>LP</td>
<td>LP</td>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>RP</td>
<td>RP</td>
<td>RP</td>
<td>RP</td>
</tr>
<tr>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
</tr>
</tbody>
</table>
Consider the simplest parentheses grammar

1. Goal ::= List
2. List ::= Pair List
   | ε
3. Pair ::= LP List RP

**FIRST** Sets

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Initial</th>
<th>1st</th>
<th>2nd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>∅</td>
<td>LP, ε</td>
<td>LP, ε</td>
</tr>
<tr>
<td>List</td>
<td>∅</td>
<td>LP, ε</td>
<td>LP, ε</td>
</tr>
<tr>
<td>Pair</td>
<td>∅</td>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>LP</td>
<td>LP</td>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>RP</td>
<td>RP</td>
<td>RP</td>
<td>RP</td>
</tr>
<tr>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
</tr>
</tbody>
</table>

- Iteration 1 adds LP to **FIRST**(Pair) and LP, ε to **FIRST**(List) and **FIRST**(Goal)
- If we take them in rule order 4, 3, 2, 1
- Algorithm reaches fixed point
FOLLOW(A):

For $A \in \text{NT}$, define $\text{FOLLOW}(A)$ as the set of tokens that can occur immediately after $A$ in a valid sentential form.

\text{FOLLOW} set is defined over the set of non-terminal symbols, \text{NT}.
To Build FOLLOW(X) for non-terminal X:

- Place EOF in FOLLOW(<start>)
- For each X as a non-terminal, initialize FOLLOW(X) to ∅

**Iterate until** no more terminals can be added to any FOLLOW(X):

  For each rule \( p \) in the grammar
  If \( p \) is of the form \( A ::= \alpha B \beta \), then
    if \( \varepsilon \in \text{FIRST}(\beta) \)
      Place \( \{\text{FIRST}(\beta) - \varepsilon, \text{FOLLOW}(A)\} \) in FOLLOW(B)
    else
      Place \( \{\text{FIRST}(\beta)\} \) in FOLLOW(B)
  If \( p \) is of the form \( A ::= \alpha B \), then
  Place FOLLOW(A) in FOLLOW(B)

**End iterate**
Computing *FOLLOW* Sets

for each \( A \in NT \)

\[
FOLLOW(A) \leftarrow \emptyset
\]

\[
FOLLOW(S) \leftarrow \{ \text{EOF} \}
\]

while (*FOLLOW* sets are still changing) do

for each \( p \in P \), of the form \( A \rightarrow B_1B_2\ldots B_k \) do

\[
\text{TRAILER} \leftarrow FOLLOW(A)
\]

for \( i \leftarrow k \) down to 1

if \( B_i \in NT \) then  \hspace{1cm} // domain checking

\[
FOLLOW(B_i) \leftarrow FOLLOW(B_i) \cup \text{TRAILER}
\]

if \( \varepsilon \in FIRST(B_i) \)  \hspace{1cm} // add right context

\[
\text{TRAILER} \leftarrow \text{TRAILER} \cup (FIRST(B_i) - \{ \varepsilon \})
\]

else \( \text{TRAILER} \leftarrow FIRST(B_i) \)  \hspace{1cm} // no \( \varepsilon \Rightarrow \) truncate the right context

else \( \text{TRAILER} \leftarrow \{ B_i \} \)  \hspace{1cm} // \( B_i \in T \Rightarrow \) only 1 symbol

To build *FOLLOW* sets, we need *FIRST* sets

Don’t add \( \varepsilon \)
Computing \textit{FOLLOW} Sets

For a production $A \rightarrow B_1B_2 \ldots B_k$:

- It works its way backward through the production: $B_k, B_{k-1}, \ldots B_1$
- It builds the \textit{FOLLOW} sets for the rhs symbols, $B_1, B_2, \ldots B_k$, not $A$
- In the absence of $\varepsilon$, $\text{FOLLOW}(B_i)$ is just $\text{FIRST}(B_{i+1})$
  - As always, $\varepsilon$ makes the algorithm more complex

To handle $\varepsilon$, the algorithm keeps track of the first word in the trailing right context as it works its way back through rhs: $B_k, B_{k-1}, \ldots B_1$
Consider the simplest parentheses grammar

Goal ::= List
List ::= Pair List |
ε
Pair ::= LP List RP

Initial Values:

- Goal, List and Pair are set to $\emptyset$
- Goal is then set to $\{ \text{EOF} \}$
Consider the simplest parentheses grammar

1. Goal ::= List
2. List ::= Pair List | ε
3. Pair ::= LP List RP

Iteration 1:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Initial</th>
<th>1st</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>EOF</td>
<td></td>
</tr>
<tr>
<td>List</td>
<td>ø</td>
<td></td>
</tr>
<tr>
<td>Pair</td>
<td>ø</td>
<td></td>
</tr>
</tbody>
</table>

If we visit the rules in order 1, 2, 3, 4

<table>
<thead>
<tr>
<th>Symbol</th>
<th>FIRST Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>LP, ε</td>
</tr>
<tr>
<td>List</td>
<td>LP, ε</td>
</tr>
<tr>
<td>Pair</td>
<td>LP</td>
</tr>
<tr>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>RP</td>
<td>RP</td>
</tr>
<tr>
<td>EOF</td>
<td>EOF</td>
</tr>
</tbody>
</table>
An Example

Consider the simplest parentheses grammar

1. Goal ::= List
2. List ::= Pair List |
3. ε
4. Pair ::= LP List RP

Iteration 1:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Initial</th>
<th>1st</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>EOF</td>
<td>EOF</td>
</tr>
<tr>
<td>List</td>
<td>∅</td>
<td>EOF</td>
</tr>
<tr>
<td>Pair</td>
<td>∅</td>
<td></td>
</tr>
</tbody>
</table>

If we visit the rules in order 1, 2, 3, 4

<table>
<thead>
<tr>
<th>Symbol</th>
<th>FIRST Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>LP, ε</td>
</tr>
<tr>
<td>List</td>
<td>LP, ε</td>
</tr>
<tr>
<td>Pair</td>
<td>LP</td>
</tr>
<tr>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>RP</td>
<td>RP</td>
</tr>
<tr>
<td>EOF</td>
<td>EOF</td>
</tr>
</tbody>
</table>
An Example

Consider the simplest parentheses grammar

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Initial</th>
<th>1st</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>EOF</td>
<td>EOF</td>
</tr>
<tr>
<td>List</td>
<td>∅</td>
<td>EOF</td>
</tr>
<tr>
<td>Pair</td>
<td>∅</td>
<td>EOF, LP</td>
</tr>
</tbody>
</table>

Iteration 1:

If we visit the rules in order 1, 2, 3, 4

<table>
<thead>
<tr>
<th>Symbol</th>
<th>FIRST Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>LP, ε</td>
</tr>
<tr>
<td>List</td>
<td>LP, ε</td>
</tr>
<tr>
<td>Pair</td>
<td>LP</td>
</tr>
<tr>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>RP</td>
<td>RP</td>
</tr>
<tr>
<td>EOF</td>
<td>EOF</td>
</tr>
</tbody>
</table>
Consider the simplest parentheses grammar

<table>
<thead>
<tr>
<th></th>
<th>Symbol</th>
<th>Initial</th>
<th>1st</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Goal</td>
<td>EOF</td>
<td>EOF</td>
</tr>
<tr>
<td>2</td>
<td>List</td>
<td>∅</td>
<td>EOF, RP</td>
</tr>
<tr>
<td>3</td>
<td>Pair</td>
<td>∅</td>
<td>EOF, LP</td>
</tr>
</tbody>
</table>

Iteration 1:

If we visit the rules in order 1, 2, 3, 4

<table>
<thead>
<tr>
<th>Symbol</th>
<th>FIRST Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>LP, ε</td>
</tr>
<tr>
<td>List</td>
<td>LP, ε</td>
</tr>
<tr>
<td>Pair</td>
<td>LP</td>
</tr>
<tr>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>RP</td>
<td>RP</td>
</tr>
<tr>
<td>EOF</td>
<td>EOF</td>
</tr>
</tbody>
</table>
Consider the simplest parentheses grammar

1. Goal ::= List
2. List ::= Pair List
3. | \( \varepsilon \)
4. Pair ::= LP List RP

**Iteration 1:**

If we visit the rules in order 1, 2, 3, 4

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Initial</th>
<th>1st</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>EOF</td>
<td>EOF</td>
</tr>
<tr>
<td>List</td>
<td>( \emptyset )</td>
<td>EOF, RP</td>
</tr>
<tr>
<td>Pair</td>
<td>( \emptyset )</td>
<td>EOF, LP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>( \text{FIRST} ) Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>LP, ( \varepsilon )</td>
</tr>
<tr>
<td>List</td>
<td>LP, ( \varepsilon )</td>
</tr>
<tr>
<td>Pair</td>
<td>LP</td>
</tr>
<tr>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>RP</td>
<td>RP</td>
</tr>
<tr>
<td>EOF</td>
<td>EOF</td>
</tr>
</tbody>
</table>
Consider the simplest parentheses grammar

1. Goal ::= List
2. List ::= Pair List | ε
3. Pair ::= LP List RP

Iteration 2:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Initial</th>
<th>1st</th>
<th>2nd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>EOF</td>
<td>EOF</td>
<td></td>
</tr>
<tr>
<td>List</td>
<td>∅</td>
<td>EOF, RP</td>
<td></td>
</tr>
<tr>
<td>Pair</td>
<td>∅</td>
<td>EOF, LP</td>
<td></td>
</tr>
</tbody>
</table>

If we visit the rules in order 1, 2, 3, 4:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>FIRST Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>LP, ε</td>
</tr>
<tr>
<td>List</td>
<td>LP, ε</td>
</tr>
<tr>
<td>Pair</td>
<td>LP</td>
</tr>
<tr>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>RP</td>
<td>RP</td>
</tr>
<tr>
<td>EOF</td>
<td>EOF</td>
</tr>
</tbody>
</table>
An Example

Consider the simplest parentheses grammar

<table>
<thead>
<tr>
<th></th>
<th>Symbol</th>
<th>Initial</th>
<th>1\textsuperscript{st}</th>
<th>2\textsuperscript{nd}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Goal</td>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
</tr>
<tr>
<td>2</td>
<td>List</td>
<td>$\varnothing$</td>
<td>EOF, RP</td>
<td>EOF, RP</td>
</tr>
<tr>
<td>3</td>
<td>Pair</td>
<td>$\varnothing$</td>
<td>EOF, LP</td>
<td></td>
</tr>
</tbody>
</table>

**Iteration 2:**

If we visit the rules in order 1, 2, 3, 4

<table>
<thead>
<tr>
<th>Symbol</th>
<th>\textit{FIRST} Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>{LP, $\varepsilon$}</td>
</tr>
<tr>
<td>List</td>
<td>{LP, $\varepsilon$}</td>
</tr>
<tr>
<td>Pair</td>
<td>{LP}</td>
</tr>
<tr>
<td>LP</td>
<td>{LP}</td>
</tr>
<tr>
<td>RP</td>
<td>{RP}</td>
</tr>
<tr>
<td>EOF</td>
<td>{EOF}</td>
</tr>
</tbody>
</table>
An Example

Consider the simplest parentheses grammar

<table>
<thead>
<tr>
<th></th>
<th>Symbol</th>
<th>Initial</th>
<th>1st</th>
<th>2nd</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Goal</td>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
</tr>
<tr>
<td>2</td>
<td>List</td>
<td>∅</td>
<td>EOF, RP</td>
<td>EOF, RP</td>
</tr>
<tr>
<td>3</td>
<td>ε</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Pair</td>
<td>∅</td>
<td>EOF, LP</td>
<td>EOF, RP, LP</td>
</tr>
</tbody>
</table>

Iteration 2:

If we visit the rules in order 1, 2, 3, 4

<table>
<thead>
<tr>
<th>Symbol</th>
<th>FIRST Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>LP, ε</td>
</tr>
<tr>
<td>List</td>
<td>LP, ε</td>
</tr>
<tr>
<td>Pair</td>
<td>LP</td>
</tr>
<tr>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>RP</td>
<td>RP</td>
</tr>
<tr>
<td>EOF</td>
<td>EOF</td>
</tr>
</tbody>
</table>
Consider the simplest parentheses grammar

1. Goal ::= List
2. List ::= Pair List
   |  ε
3. Pair ::= LP List RP

Iteration 2:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Initial</th>
<th>1&lt;sup&gt;st&lt;/sup&gt;</th>
<th>2&lt;sup&gt;nd&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
</tr>
<tr>
<td>List</td>
<td>∅</td>
<td>EOF, RP</td>
<td>EOF, RP</td>
</tr>
<tr>
<td>Pair</td>
<td>∅</td>
<td>EOF, LP</td>
<td>EOF, RP, LP</td>
</tr>
</tbody>
</table>

If we visit the rules in order 1, 2, 3, 4

<table>
<thead>
<tr>
<th>Symbol</th>
<th>FIRST Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>LP, ε</td>
</tr>
<tr>
<td>List</td>
<td>LP, ε</td>
</tr>
<tr>
<td>Pair</td>
<td>LP</td>
</tr>
<tr>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>RP</td>
<td>RP</td>
</tr>
<tr>
<td>EOF</td>
<td>EOF</td>
</tr>
</tbody>
</table>
An Example

Consider the simplest parentheses grammar

1. Goal ::= List
2. List ::= Pair List
   | ε
3. Pair ::= LP List RP

Iteration 2:

- Production 1 adds nothing new
- Production 2 adds RP to FOLLOW(Pair) from FOLLOW(List), ε ∈ FIRST(List)
- Production 3 does nothing
- Production 4 adds nothing new

Iteration 3 produces the same result ⇒ reached a fixed point

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Initial</th>
<th>1st</th>
<th>2nd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>EOF</td>
<td>EOF</td>
<td>EOF</td>
</tr>
<tr>
<td>List</td>
<td>Ø</td>
<td>EOF, RP</td>
<td>EOF, RP</td>
</tr>
<tr>
<td>Pair</td>
<td>Ø</td>
<td>EOF, LP</td>
<td>EOF, RP, LP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>FIRST Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>LP, ε</td>
</tr>
<tr>
<td>List</td>
<td>LP, ε</td>
</tr>
<tr>
<td>Pair</td>
<td>LP</td>
</tr>
<tr>
<td>LP</td>
<td>LP</td>
</tr>
<tr>
<td>RP</td>
<td>RP</td>
</tr>
<tr>
<td>EOF</td>
<td>EOF</td>
</tr>
</tbody>
</table>
Building Top-down Parsers

Building the $FIRST^+$ set

- Need a $FIRST^+$ set for every rule

<table>
<thead>
<tr>
<th>Symbol</th>
<th>$FIRST$</th>
<th>$FOLLOW$</th>
<th>Rule</th>
<th>$FIRST^+$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>LP, ε</td>
<td>EOF</td>
<td>1</td>
<td>EOF, LP</td>
</tr>
<tr>
<td>List</td>
<td>LP, ε</td>
<td>EOF, RP</td>
<td>2</td>
<td>LP</td>
</tr>
<tr>
<td>Pair</td>
<td>LP</td>
<td>EOF, RP, LP</td>
<td>3</td>
<td>EOF, RP</td>
</tr>
<tr>
<td>LP</td>
<td>LP</td>
<td>-</td>
<td>3</td>
<td>EOF, RP</td>
</tr>
<tr>
<td>RP</td>
<td>RP</td>
<td>-</td>
<td>3</td>
<td>EOF, RP</td>
</tr>
<tr>
<td>EOF</td>
<td>EOF</td>
<td>-</td>
<td>4</td>
<td>LP</td>
</tr>
</tbody>
</table>

Rule 1: Goal ::= List
Rule 2: List ::= Pair List
Rule 3: | ε
Rule 4: Pair ::= LP List RP
Building Top-down Parsers

Building the complete parse table

- Need a row for every **NT** and a column for every **T**
- Need an interpreter for the table (skeleton parser)

<table>
<thead>
<tr>
<th>Rule</th>
<th>FIRST+</th>
<th>LP</th>
<th>RP</th>
<th>EOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal ::= List</td>
<td>1</td>
<td>EOF, LP</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>List ::= Pair List</td>
<td>2</td>
<td>LP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pair ::= LP List RP</td>
<td>4</td>
<td>LP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Goal ::= List

List ::= Pair List

Pair ::= LP List RP

ε
Building the complete parse table

- Need a row for every `NT` and a column for every `T`
- Need an interpreter for the table (skeleton parser)

<table>
<thead>
<tr>
<th>Rule</th>
<th><code>FIRST+</code></th>
<th><code>LP</code></th>
<th><code>RP</code></th>
<th><code>EOF</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Goal ::= List</td>
<td>EOF, RP</td>
<td>Goal</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2. List ::= Pair List</td>
<td>LP</td>
<td>List</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3.</td>
<td>ε</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Pair ::= LP List RP</td>
<td>EOF, RP</td>
<td>Pair</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Building Top-down Parsers

Building the complete parse table

- Need a row for every NT and a column for every T
- Need an interpreter for the table (skeleton parser)

<table>
<thead>
<tr>
<th>Rule</th>
<th>FIRST+</th>
<th>LP</th>
<th>RP</th>
<th>EOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EOF, RP</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>LP</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>EOF, RP</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>LP</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 1       | Goal ::= List | Goal | 1 | 1 |
| 2       | List ::= Pair List | List | 2 | 3 | 3 |
| 3       |         | Pair | 4 |
| 4       | Pair ::= LP List RP |     |    |   |
Building the complete parse table

- Need a row for every **NT** and a column for every **T**
- Need an interpreter for the table (skeleton parser)

<table>
<thead>
<tr>
<th>Rule</th>
<th>FIRST+</th>
<th>LP</th>
<th>RP</th>
<th>EOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Goal ::= List</td>
<td>EOF, RP</td>
<td>Goal</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2 List ::= Pair List</td>
<td>LP</td>
<td>List</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>ε</td>
<td>Pair</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4 Pair ::= LP List RP</td>
<td>LP</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
Input: a string \( w \) and a parsing table \( M \) for \( G \)

```
push eof
push Start Symbol
token ← next_token()
X ← top-of-stack
repeat
    if X is a terminal then
        if X == token then
            pop X
token ← next_token()
        else error()
    else /* X is a non-terminal */
        if \( M[X, \text{token}] = X \rightarrow Y_1 Y_2 \ldots Y_k \) then
            pop X
            push \( Y_k, Y_{k-1}, \ldots, Y_1 \)
        else error()
        X ← top-of-stack
    until X = EOF
if token != EOF then error()
```

\( M \) is the parse table
Things to do:

• Start programming in C.
• Read Scott, Chapter 3.1 - 3.3; ALSU 7.1
• Read Scott, Chapter 8.1 - 8.2; ALSU 7.1 - 7.3