1 LL Parsing Ambiguities

An LL(k) grammar is a CFG used by a parser that scans input left-to-right (“L”), leftmost derivation (“L”), and uses k tokens of lookahead to predict the correct production. We’ve previously seen that a grammar is ambiguous if it has a parse tree that is not unique. A more formal definition of LL conflicts uses FIRST and FOLLOW sets.

- **FIRST(A)**: the set of all terminals that could occur first in an expansion of the terminal or nonterminal A (include $\epsilon$ if A can expand to $\epsilon$)
- **FOLLOW(A)**: the set of all terminals that could follow an occurrence of the terminal or nonterminal A in a (partial) derivation

There are two main types of LL(1) conflicts:

- **FIRST/FIRST**: The FIRST sets of two different productions for same non-terminal intersect.
- **FIRST/FOLLOW**: The FIRST set of a grammar rule contains an epsilon and the intersection with its FOLLOW set is not empty.

**Ex.1** Are the following grammars LL(1)? Justify your answer using FIRST and FOLLOW sets. (Thanks to Karen Lemone, at WPI, for these problems.)

(a) \[ A \rightarrow dA \mid dB \mid f \quad B \rightarrow g \]

**Answer:** This is an instance of a FIRST/FIRST conflict. We can’t tell which of A’s production rule to follow. The FIRST and FOLLOW sets for this grammar are:

\[
\begin{align*}
\text{FIRST}(dA) &= \{ 'd' \} \\
\text{FIRST}(dB) &= \{ 'd' \} \\
\text{FIRST}(f) &= \{ 'f' \} \\
\text{FIRST}(g) &= \{ 'g' \}
\end{align*}
\]

\[
\begin{align*}
\text{FOLLOW}(A) &= \{ \} \\
\text{FOLLOW}(B) &= \{ \}
\end{align*}
\]

(b) \[ A \rightarrow B \mid A + A \mid A \ast A \mid (A) \quad B \rightarrow \text{Num} \mid \text{Id} \]

**Answer:** This is also an instance of a FIRST/FIRST conflict that is caused by left recursion. The FIRST and FOLLOW sets for this grammar are:

\[
\begin{align*}
\text{FIRST}(\text{Num}) &= \{ \text{Num} \} \\
\text{FIRST}(\text{Id}) &= \{ \text{Id} \} \\
\text{FIRST}(\text{A } \ast \text{ A}) &= \{ '(' \}
\end{align*}
\]

\[
\begin{align*}
\text{FIRST}(A \ast A) &= \{ \text{Id, Num, '}' \} \\
\text{FIRST}(A + A) &= \{ \text{Id, Num, '}' \}
\end{align*}
\]

\[
\begin{align*}
\text{FOLLOW}(A) &= \{ '+', '*', ']' \} \\
\text{FOLLOW}(B) &= \{ '+', '*', ']' \}
\end{align*}
\]
Answer: This is an instance of a FIRST/FOLLOW conflict. FIRST(X) contains the empty string and the intersection of FIRST(X) and FOLLOW(X) is not empty:

\[
\begin{align*}
\text{FIRST}(Xd) &= \{ 'd' \} \\
\text{FIRST}(C) &= \{ \epsilon \} \\
\text{FIRST}(B a) &= \{ 'd' \}
\end{align*}
\]

\[
\begin{align*}
\text{FIRST}(\epsilon) &= \{ \epsilon \} \\
\text{FIRST}(d) &= \{ 'd' \}
\end{align*}
\]

\[
\begin{align*}
\text{FOLLOW}(S) &= \{ \} \\
\text{FOLLOW}(X) &= \{ 'd' \} \\
\text{FOLLOW}(C) &= \{ 'd' \} \\
\text{FOLLOW}(B) &= \{ 'a' \}
\end{align*}
\]

2 Resolving Conflicts

Ex.2 For the grammar from Ex.1(a), rewrite the grammar so that it is LL(1) by introducing the non-terminal \( AB \rightarrow A \mid B \).

Answer: This technique of factoring out terminals on the left decreases the amount of lookahead you need to perform.

\[
\begin{align*}
A &: d \ AB \\
B &: g \\
AB &: A \\
&| f \\
&| B
\end{align*}
\]

Ex.3 (Challenge Question) Consider the following grammar for numerical expressions with division, addition, and unary minus: \( E \rightarrow \text{Num} \mid E/E \mid E + E \mid -E \)

(a) Rewrite the grammar so that it is LL(1), so that ‘/’ has higher precedence than ‘+’, and so that ‘-’ has highest precedence. (Note that ‘+’ and ‘/’ will be parsed in a right-associative way. We can fix ‘+’ and ‘/’ to be left-associative in the semantic actions.) You may find the following procedure helpful for removing left recursion:

- Given a production \( A \rightarrow A\alpha_1 \mid ... \mid A\alpha_n \mid \beta_1 \mid ... \mid \beta_m \):
  - Replace the A production with \( A \rightarrow \beta_1 A' \mid ... \mid \beta_m A' \)
  - Create a new non-terminal \( A' \rightarrow \epsilon \mid \alpha_1 A' \mid ... \mid \alpha_n A' \)

Answer:

\[
\begin{align*}
\text{expr} &: \text{expr1 rest} \\
\text{expr1} &: \text{expr2 rest1} \\
\text{expr2} &: ' - ' expr2 \\
\text{rest} &: \epsilon \\
\text{rest1} &: \epsilon \\
&| '+' expr \\
&| '/' expr
\end{align*}
\]

(b) Compute the FIRST and FOLLOW sets for your re-written LL(1) grammar.

Answer:

\[
\begin{align*}
\text{FIRST}(\text{expr1 rest}) &= \text{FIRST}(\text{expr2 rest1}) = \{ ' - ', \text{NUM} \}
\end{align*}
\]

\[
\begin{align*}
\text{FIRST}('+') &= \{ '+' \} \\
\text{FIRST}('/') &= \{ '/' \}
\end{align*}
\]

\[
\begin{align*}
\text{FIRST}('-') &= \{ '-' \} \\
\text{FIRST}(\text{NUM}) &= \{ \text{NUM} \}
\end{align*}
\]

\[
\begin{align*}
\text{FIRST}(\epsilon) &= \{ \epsilon \}
\end{align*}
\]
FOLLOW(expr2) = \{'/','+','\}'
FOLLOW(expr1) = FOLLOW(rest1) = \{'+','-\}'
FOLLOW(expr) = FOLLOW(rest) = \{\}'

(c) Draw the LL(1) parsing table for the grammar. You may need following rules:

- For each production \(X \rightarrow A_1...A_n\):
  - For each \(1 \leq i \leq n\), and for each \(b \) in First\((A_i)\): Set \(T[X,b] = X \rightarrow A_1...A_n\).
    Stop when \(\epsilon\) is not in First\((A_i)\).
  - If \(A_1...A_n \rightarrow \ast \epsilon\), then for each \(b \) in Follow\((X)\): Set \(T[X,b] = \epsilon\).

Answer:

<table>
<thead>
<tr>
<th></th>
<th>(-)</th>
<th>(\text{NUM})</th>
<th>(\slash)</th>
<th>(\text{+})</th>
<th>(-)</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>expr (\rightarrow) expr1 rest</td>
<td>expr (\rightarrow) expr1 rest</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rest</td>
<td>expr1 (\rightarrow) expr2 rest1</td>
<td>expr1 (\rightarrow) expr2 rest1</td>
<td>rest (\rightarrow) + expr</td>
<td>(\epsilon)</td>
<td></td>
</tr>
<tr>
<td>expr1</td>
<td>expr1 (\rightarrow) expr2 rest1</td>
<td>expr1 (\rightarrow) expr2 rest1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rest1</td>
<td>rest1 (\rightarrow) / expr1</td>
<td>rest1 (\rightarrow) / expr1</td>
<td>(\epsilon)</td>
<td>(\epsilon)</td>
<td></td>
</tr>
<tr>
<td>expr2</td>
<td>expr2 (\rightarrow) - expr2</td>
<td>expr2 (\rightarrow) NUM</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(d) (Challenge Question) Find a context-free language (CFL) for which there exists no LL\((k)\) grammar, for any \(k\).

(e) (Challenge Question) Find a CFL for which there is an LL\((k)\) grammar, for some \(k > 1\), but no LL\((1)\) grammar.