Eliminating Immediate Left Recursion

Left recursive productions can cause recursive descent parsers to loop forever. Therefore, we consider how to eliminate left recursion from a grammar.

Consider the productions $A \rightarrow A\alpha \mid \beta$ where $\alpha$ and $\beta$ are sequences of terminals and nonterminals that do not start with $A$. These productions can be used to generate the following strings:

$$\beta \quad \beta\alpha \quad \beta\alpha\alpha \quad \beta\alpha\alpha\alpha \quad \text{etc.}$$

Note that the same language can be generated by the productions

$$A \rightarrow \beta \ R$$
$$R \rightarrow \alpha \ R \mid \epsilon$$

where $R$ is a new nonterminal. Note that the $R$-production is right recursive, which implies that we might have altered the associativity of an operator. We will discuss how to handle this possibility later.

In general, immediate left recursion (as we have above) may be removed as follows. Suppose we have the $A$-productions

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$$

where no $\beta_i$ begins with $A$. We replace the $A$-productions by

$$A \rightarrow \beta_1A' \mid \beta_2A' \mid \ldots \mid \beta_mA'$$
$$A' \rightarrow \alpha_1A' \mid \alpha_2A' \mid \ldots \mid \alpha_nA' \mid \epsilon$$

where $A'$ is a new nonterminal.
Let's eliminate left recursion from the grammar below (note accompanying parse tree for \( a + a + a \)):

\[
E \rightarrow E + T \\
\quad \mid T \\
T \rightarrow T * F \\
\quad \mid F \\
F \rightarrow ( E ) \\
\quad \mid a
\]

Note how the parse tree grows down toward the left, indicating the left associativity of \( + \).

Eliminating left recursion we get the following grammar. Note parse tree for \( a + a + a \):

\[
E \rightarrow TE' \\
E' \rightarrow +TE' \mid \varepsilon \\
T \rightarrow FT' \\
T' \rightarrow *FT' \mid \varepsilon \\
F \rightarrow ( E ) \mid a
\]

Note how the parse tree grows down toward the right, indicating that operator \( + \) is now right associative.
Algorithm for Eliminating General Left Recursion

Arrange nonterminals in some order $A_1, A_2, \ldots, A_n$.

\begin{verbatim}
for $i := 1$ to $n$ do begin
    for $j := 1$ to $i - 1$ do begin
        Replace each production of the form $A_i \rightarrow A_j \beta$ by the productions:
        
        $A_i \rightarrow \alpha_1 \beta | \alpha_2 \beta | \ldots | \alpha_k \beta$
        
        where
        
        $A_j \rightarrow \alpha_1 | \alpha_2 | \ldots | \alpha_k$

        are all the current $A_j$ productions.
    end { for j }

    Remove immediate left recursion from the $A_i$ productions, if necessary.
end { for i }
\end{verbatim}

Example: $S \rightarrow Aa | b$

$A \rightarrow Ac | Sd | \varepsilon$

- Let's use the ordering $S, A$ ($S = A_1, A = A_2$).
- When $i = 1$, we skip the "for j" loop and remove immediate left recursion from the $S$ productions (there is none).
- When $i = 2$ and $j = 1$, we substitute the $S$-productions in $A \rightarrow Sd$ to obtain the $A$-productions

  $A \rightarrow Ac | Aad | bd | \varepsilon$

- Eliminating immediate left recursion from the $A$ productions yields the grammar:

  $S \rightarrow Aa | b$
  $A \rightarrow bdA' | A'$
  $A' \rightarrow cA' | adA' | \varepsilon$
Left Factoring

Left factoring is a grammar transformation that is useful for producing a grammar suitable for top-down parsing. The basic idea is that when it is not clear which of two alternative productions to use to expand a nonterminal A, we may be able to rewrite the A-productions to defer the decision until we have seen enough of the input to make the right choice.

To illustrate, consider the productions

\[
S \rightarrow \texttt{if } E \texttt{ then } S \\
| \quad \texttt{if } E \texttt{ then } S \texttt{ else } S
\]

on seeing the input token \texttt{if}, we cannot immediately tell which production to choose to expand S.

In general, if \( A \rightarrow \alpha \beta_1 \mid \alpha \beta_2 \) are two A-productions, and the input begins with a nonempty string derived from \( \alpha \), we do not know whether to expand to \( \alpha \beta_1 \) or to \( \alpha \beta_2 \). Instead, the grammar may be changed. The formal technique is to change

\[
A \rightarrow \alpha \beta_1 \mid \alpha \beta_2
\]

to

\[
A \rightarrow \alpha A' \\
A' \rightarrow \beta_1 \mid \beta_2
\]

Thus, we can rewrite the grammar for if-statement as:

\[
S \rightarrow \texttt{if } E \texttt{ then } S \texttt{ ElsePart} \\
\texttt{ElsePart} \rightarrow \texttt{else } S \mid \varepsilon
\]