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 α and β are sequences of terminals and nonterminals that do not start with A. These productions can be used to generate the following strings:

$\beta \qquad \beta\alpha \qquad \beta\alpha\alpha \qquad \beta\alpha\alpha\alpha \qquad \beta\alpha\alpha\alpha\alpha \qquad$ etc.

Note that the same language can be generated by the productions

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

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

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

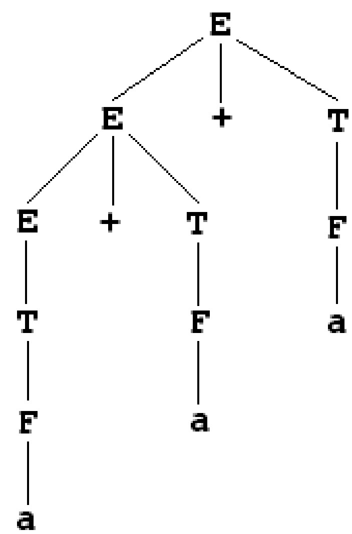
where no β_i begins with A. We replace the A-productions by

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

where A' is a new nonterminal.

Let's eliminate left recursion from the grammar below (note accompanying parse tree for **a + a + a**):

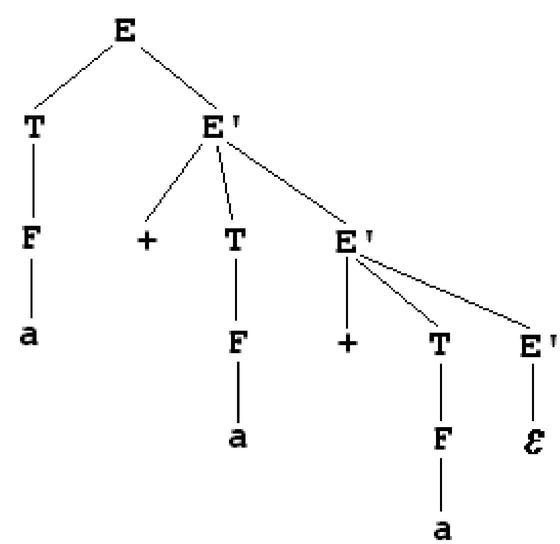
$$
\begin{aligned}
E &\rightarrow E \;\mathbf{+}\; T \\
  &\;\;|\;\; T \\
T &\rightarrow T \;\mathbf{*}\; F \\
  &\;\;|\;\; F \\
F &\rightarrow \mathbf{(}\; E \;\mathbf{)} \\
  &\;\;|\;\; \mathbf{a}
\end{aligned}
$$



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**:

$$
\begin{aligned}
E &\rightarrow TE' \\
E' &\rightarrow +TE' \mid \varepsilon \\
T &\rightarrow FT' \\
T' &\rightarrow *FT' \mid \varepsilon \\
F &\rightarrow (\ E\ ) \mid a
\end{aligned}
$$



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 $A_1, A_2, \ldots, A_n$.

for *i* := 1 **to** *n* **do begin**
  for *j* := 1 **to** i - 1 **do begin**
    Replace each production of the form $A_i \to A_j\beta$ by the productions:

$$A_i \to \alpha_1\beta \mid \alpha_2\beta \mid \ldots \mid \alpha_k\beta$$

    where

$$A_j \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_k$$

    are all the current A_j productions.

  end { for j }

  Remove immediate left recursion from the A_i productions, if necessary.

end { for i }

Example: $S \to Aa \mid b$
$A \to Ac \mid Sd \mid \varepsilon$

- 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 $A \to Sd$ to obtain the A-productions

$$A \to Ac \mid \underline{Aa}d \mid \underline{b}d \mid \varepsilon$$

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

$$S \to Aa \mid b$$
$$A \to bdA' \mid A'$$
$$A' \to cA' \mid adA' \mid \varepsilon$$

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

$$
\begin{aligned}
S &\rightarrow \textbf{if } E \textbf{ then } S \\
&\mid \textbf{ if } E \textbf{ then } S \textbf{ else } S
\end{aligned}
$$

on seeing the input token **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 α, 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

$$
\begin{aligned}
A &\rightarrow \alpha A' \\
A' &\rightarrow \beta_1 \mid \beta_2
\end{aligned}
$$

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

$$
\begin{aligned}
S &\rightarrow \textbf{if } E \textbf{ then } S \; \mathit{ElsePart} \\
\mathit{ElsePart} &\rightarrow \textbf{else } S \mid \varepsilon
\end{aligned}
$$