Elimination of Left Recursion

Def: A grammar is left recursive if \( \exists \) a non-terminal \( A \) such that

\[
A \Rightarrow^* A_\alpha
\]

Aside: Let's define a binary relation called \( r \). We say that

\[
V_i \, r \, V_j \quad \text{if} \quad V_i \rightarrow V_j \alpha \in P.
\]

Now let's compute the transitive closure of \( r \), \( r^* \). If \( \exists \) a \( V_i \supset V_i \, r^* \, V_i \) holds, then the grammar is left recursive.

An algorithm that eliminates immediate left recursion is

1. Order the non-terminals yielding

\[
V_n = \{ A_1, A_2, \ldots, A_n \}
\]

2. For each \( A_i \), Let

\[
A_i \rightarrow A_{i,1} \alpha_1
\]

\[
| A_{i,2} \alpha_2
\]

\[
| \ldots
\]

\[
| A_{i,m} \alpha_m
\]

\[
| \beta_1
\]

\[
| \beta_2
\]

\[
| \ldots
\]

\[
| \beta_n
\]

\( \exists \) no \( \beta_j \) begins with an \( A_i \). Replace these \( A_i \) productions with

\[
A_i \rightarrow \beta_1 | \beta_2 | \ldots | \beta_n | \beta_1 A_i' | \beta_2 A_i' | \ldots | \beta_n A_i'
\]

and add the productions

\[
A_i' \rightarrow \alpha_1 | \alpha_2 | \ldots | \alpha_m | \alpha_1 A_i' | \alpha_2 A_i' | \ldots | \alpha_m A_i'
\]

where \( A_i' \) is a new non-terminal symbol.

Let's consider the following grammar

\[
\begin{align*}
E & \rightarrow E + T \\
& | T \\
T & \rightarrow T * P \\
& | P \\
P & \rightarrow (E) \\
& | V
\end{align*}
\]
If we eliminate left recursion, we get

\[ V_n = \{ \} \]

Click here for the answer.

We have

\[
E \rightarrow E + T \\
| T
\]

\[
A_i \rightarrow A_i \; \alpha_i \\
| \beta_i
\]

The transformation rules are

\[
A_i \rightarrow \beta_i \\
| \beta_i \; A_i'
\]

\[
A_i' \rightarrow \alpha_i \\
| \alpha_i \; A_i'
\]

\[
E \rightarrow T \\
| T \; E'
\]

\[
E' \rightarrow + T \\
| + T \; E'
\]

\[
T \rightarrow P \\
| P \; T'
\]

\[
T' \rightarrow * P \\
| * P \; T'
\]

\[
P \rightarrow ( \; E \; ) \\
| V
\]

Note: \( L(G) \equiv L(G') \)
ANSWERS

From the above example we get

\[ V_n = \{ E, T, P \} \]