University of Asia Pacific

Department of Computer Science & Engineering

CSE 430: Compiler Design

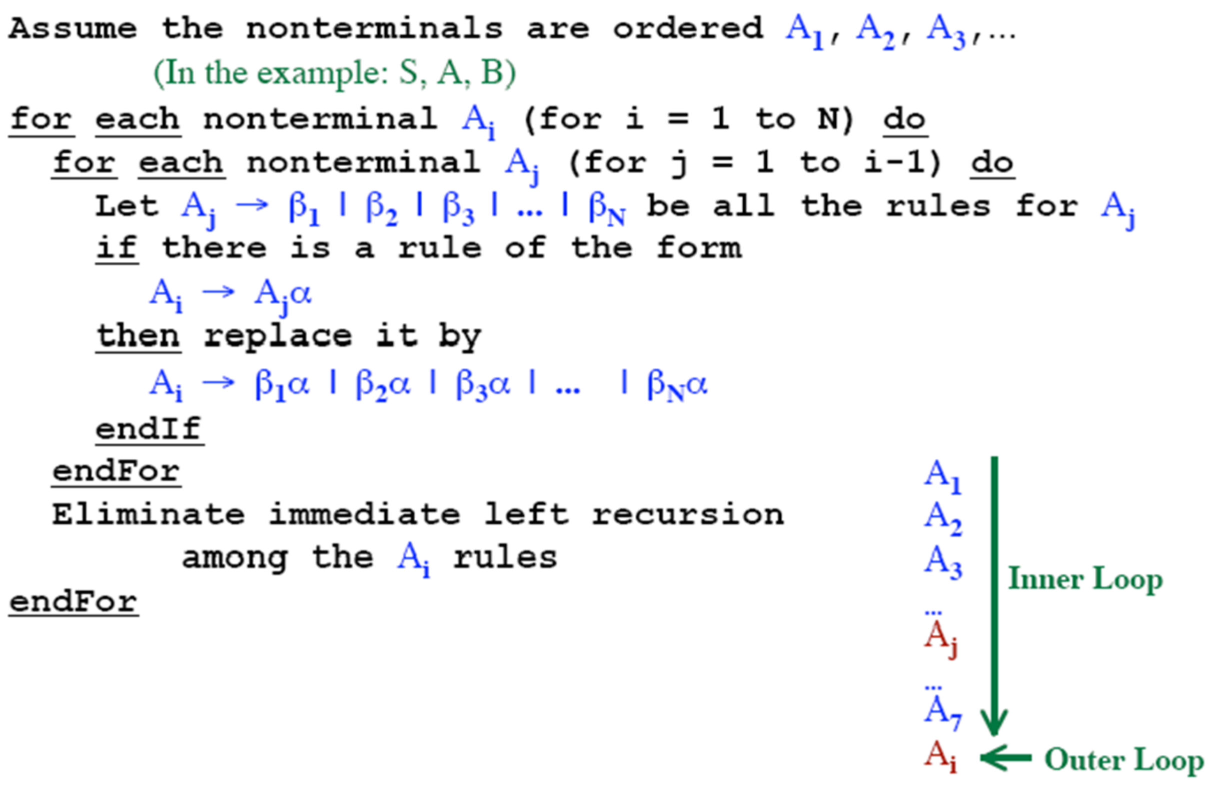
Lab 4: Elimination of Left Recursion in a grammar

We have already seen what is left recursion in theory class and we know how to eliminate left recursion in a grammar. Whenever there is a production in the form of:

$$A \rightarrow A\alpha$$

It is considered as left recursion.
There are two types of left recursion:
i) Immediate left recursion
ii) Non-immediate left recursion

The algorithm for eliminating left recursion is given below:



```
Assume the nonterminals are ordered $A_1$, $A_2$, $A_3$,...
        (In the example: S, A, B)
for each nonterminal $A_i$ (for i = 1 to N) do
  for each nonterminal $A_j$ (for j = 1 to i-1) do
    Let $A_j \rightarrow \beta_1 \mid \beta_2 \mid \beta_3 \mid \ldots \mid \beta_N$ be all the rules for $A_j$
    if there is a rule of the form
      $A_i \rightarrow A_j\alpha$
    then replace it by
      $A_i \rightarrow \beta_1\alpha \mid \beta_2\alpha \mid \beta_3\alpha \mid \ldots \mid \beta_N\alpha$
    endIf
  endFor
  Eliminate immediate left recursion
          among the $A_i$ rules
endFor
```

Now, use this algorithm to eliminate left recursion from a grammar taken as an input in the console. Follow the theory class lecture slides for more clarification about the topic. You can use any programming language as per your choice.

Sample Input:

```
E->E+T|T
```

Sample Output:

```
After elimination of left recursion the grammar is:
E  -> TE′
E′ -> +TE′ | ε
```

Another Sample Input:

```
T -> T*F | F
```

Sample Output:

```
After elimination of left recursion the grammar is:
T → FT′
T′ → *FT′ | ε
```