

Theory of Automata

Context Free Grammars

Week-12-Lecture-01

Hafiz Tayyeb Javed

Contents

What we have covered

- Simplification of CFGs
 - Killing Λ -Productions
 - Killing unit-productions
 - Removing Useless Variables
 - Symbols & Productions
 - Augmented Grammar

Today's Lecture

- Removal of Left Recursion
- Expression Grammar
- Left Factoring

Augmented Grammar

- Add a new start symbol $S_0 \rightarrow S$
- This change guarantees that the start symbol of the new grammar will not occur on the *rhs* of any rule.

Removal of Left Recursion

- Consider the grammar $A \rightarrow Aa \mid b$
- **Halting condition of top-down parsing** depend upon the generation of terminal prefixes to discover dead ends.
- Repeated application of above rule fail to generate a prefix that can terminate the parse.

Removing left recursion

- To remove left recursion from A , the A rules are divided into two groups.
Left recursive and others

$$A \rightarrow Au_1 \mid Au_2 \mid Au_3 \mid \dots \mid Au_j$$

$$A \rightarrow V_1 \mid V_2 \mid V_3 \mid \dots \mid V_k$$

Solution:

$$A \rightarrow V_1 \mid V_2 \mid V_3 \mid \dots \mid V_k \mid V_1Z \mid V_2Z \mid \dots \mid V_kZ$$

$$Z \rightarrow u_1Z \mid u_2Z \mid u_3Z \mid \dots \mid u_jZ \mid u_1 \mid u_2 \mid u_3 \mid \dots \mid u_j$$

Removal of Left Recursion

Or Equivalently

$$A \rightarrow Au_1 \mid Au_2 \mid Au_3 \mid \dots \mid Au_j$$

$$A \rightarrow V_1 \mid V_2 \mid V_3 \mid \dots \mid V_k$$

Solution:

$$A \rightarrow V_1Z \mid V_2Z \mid \dots \mid V_kZ$$

$$Z \rightarrow u_1Z \mid u_2Z \mid u_3Z \mid \dots \mid u_jZ \mid \lambda$$

Example

- $A \rightarrow Aa \mid b$

Solution:

$A \rightarrow bZ \mid b$

$Z \rightarrow aZ \mid a$

OR

$A \rightarrow bZ$

$Z \rightarrow aZ \mid \lambda$

- $A \rightarrow Aa \mid Ab \mid b \mid c$

$A \rightarrow bZ \mid cZ \mid b \mid c$

$Z \rightarrow aZ \mid bZ \mid a \mid b$

Consider another example

- $A \rightarrow AB \mid BA \mid a$

- $B \rightarrow b \mid c$

$A \rightarrow BAZ \mid aZ \mid BA \mid a$

$Z \rightarrow BZ \mid B$

$B \rightarrow b \mid c$

- The above transformations remove left-recursion by creating a right-recursive grammar; but this changes the associativity of our rules. Left recursion makes left associativity; right recursion makes right associativity. Example : We start out with a grammar :

$$Expr \rightarrow Expr + Term \mid Term$$

$$Term \rightarrow Term * Factor \mid Factor$$

$$Factor \rightarrow (Expr) \mid Int$$

- After having applied standard transformations to remove left-recursion, we have the following grammar :

$$Expr \rightarrow Term Expr'$$

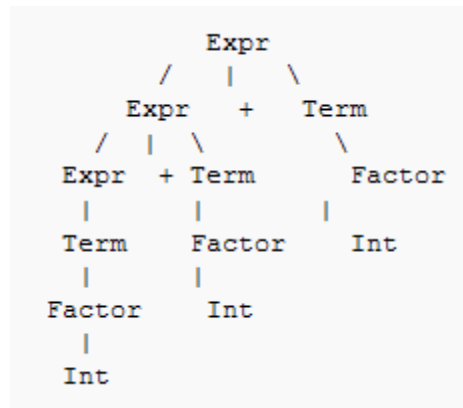
$$Expr' \rightarrow + Term Expr' \mid \epsilon$$

$$Term \rightarrow Factor Term'$$

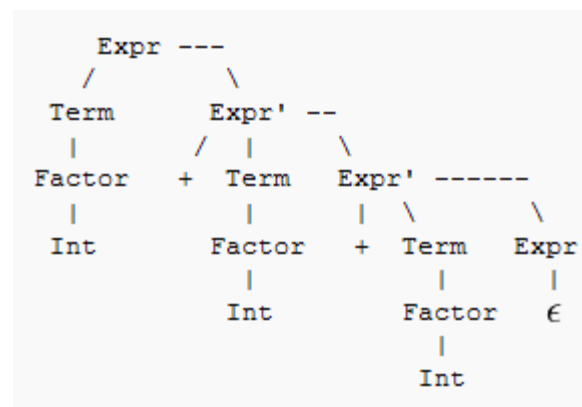
$$Term' \rightarrow * Factor Term' \mid \epsilon$$

$$Factor \rightarrow (Expr) \mid Int$$

- Parsing the string 'a + a + a' with the first grammar in an LALR parser (which can recognize left-recursive grammars) would have resulted in the parse tree:



- This parse tree grows to the left, indicating that the '+' operator is left associative, representing $(a + a) + a$.
- But now that we've changed the grammar, our parse tree looks like this :



- We can see that the tree grows to the right, representing $a + (a + a)$. We have changed the associativity of our operator '+', it is now right-associative. While this isn't a problem for the associativity of addition, it would have a significantly different value if this were subtraction.
- The problem is that normal arithmetic requires left associativity. Several solutions are: (a) rewrite the grammar to be left recursive, or (b) rewrite the grammar with more nonterminals to force the correct precedence/associativity, or (c) if using YACC or Bison, there are operator declarations, %left, %right and %nonassoc, which tell the parser generator which associativity to force.

Left Factoring

The grammar is unambiguous and non-left-recursion but we don't know which rule to apply on a symbol in:

$$\begin{aligned} S &\rightarrow VU_1 \\ &\rightarrow VU_2 \\ &\rightarrow VU_3 \\ &\dots\dots\dots \\ &\rightarrow VU_n \end{aligned}$$

However, it can be factored as follows.

$$\begin{aligned} S &\rightarrow VB \\ B &\rightarrow U_1 \\ &\rightarrow U_2 \\ &\rightarrow U_3 \\ &\dots\dots\dots \\ &\rightarrow U_n \end{aligned}$$

- Should we apply right factoring too?