Consider state q_0:

$\delta(q_0, a) = q_1$ (E)

$\delta(q_0, b) = q_3$. But, q_3 belongs to group G_2 and the representative of this group is q_1. So, instead of q_3 we can write q_1. Therefore,

$\delta(q_0, b) = q_1$ (F)

Consider state q_1:

$\delta(q_1, a) = q_2$ which is equal to q_1 since q_1 and q_2 are in the same group and q_1 is the representative of that group. So,

$\delta(q_1, a) = q_1$ (G)
$\delta(q_1, b) = q_4$ (H)

Consider the state q_4:

$\delta(q_4, a) = q_4$ (I)
$\delta(q_4, b) = q_4$ (J)

The transition table for the transitions obtained from (E) to (J) are shown in the table 4.4 and the reduced DFA using the transition graph is shown in figure 4.5. Here, q_4 is the final state in reduced DFA since it is the final state in the given DFA.

		←Σ→	
	δ	a	b
↑ States ↓	→q_0	q_1	q_1
	q_1	q_1	q_4
	(q_4)	q_4	q_4

Table 4.4 Transition table

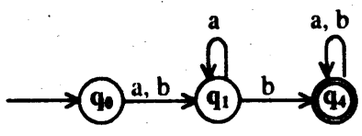


Fig. 4.5 Reduced DFA

Exercises:

1. Show that the regular languages are closed under
   a. Union b. Concatenation c. Star closure
2. Show that the regular languages are closed under complementation
3. Show that the regular languages are closed under intersection
4. Show that the regular languages are closed under difference
5. What is homomorphism? Explain with example
6. Show that the regular languages are closed under homomorphism
7. What are the languages accepted by FA and what are not accepted by FA? Specify some languages which are not accepted by FA.

8. What is Pigeonhole principle?
9. State and prove Pumping Lemma
10. What is Pumping Lemma? Why it is used?
11. What are the applications of Pumping Lemma?
12. What is the general strategy used in Pumping Lemma for proving certain languages are not regular?
13. Show that L = $\{ww^R \mid w \in (0+1)^*\}$ is not regular.
14. Show that L = $\{a^n b^n \mid n \geq 0\}$ is not regular.
15. Show that L = $\{a^n b^l \mid n \neq l\}$ is not regular.
16. Show that L = $\{a^i b^j \mid i > j\}$ is not regular.
17. Show that L = $\{a^n b^l c^{n+l} \mid n,l \geq 0\}$ is not regular.
18. Show that L = $\{a^{n!} \mid n \geq 0\}$ is not regular.
19. Show that L = $\{a^n \mid n$ is prime$\}$ is not regular.
20. Show that L = $\{ w \mid n_a(w) = n_b(w)\}$ is not regular. Is L* regular?
21. Show that L = $\{ w \mid n_a(w) < n_b(w)\}$ is not regular.
22. Show that L = $\{ ww \mid w \in \{a,b\}^* \}$ is not regular.
23. Show that L = $\{ (ab)^n a^k \mid n > k, k \geq 0\}$ is not regular.
24. Show that L = $\{ a^n \mid n = k^2$ for $k \geq 0\}$ is not regular.
25. Show that L = $\{0^n$ such that n is not a prime number$\}$ is not regular.
26. Show that L = $\{0^n 1^n 2^n$ for $n \geq 0 \}$ is not regular

5

Context Free Grammars

What we will know after reading this chapter?

- Advantages of regular and non-regular languages
- Definition of Context Free Grammar (CFG)
- To obtain CFGs for various types of context free languages
- To check whether the given languages is CFG
- Leftmost derivation
- Rightmost derivation
- Derivation Tree(Parse tree)
- Yield of a tree
- Partial derivation tree
- Ambiguous grammar
- Solution to various problems of ambiguous grammars
- Inherently ambiguous grammar
- Parsing
- Simple grammar (S-Grammar)
- To obtain S-Grammars for the specified languages
- Applications of context free grammars
- BNF notations with applications
- Solution to more than 30 problems of various nature

In the previous chapters we have seen that it is possible to construct FA to accept regular languages. But, it is not possible to construct FA to accept non-regular languages. So, there should be a machine to accept the non-regular languages also. A pushdown automaton can be constructed to accept the non-regular languages and all regular languages. In this chapter we concentrate on the definition of a context free grammar, how to obtain context free grammars, how a language can be derived from the context free grammars and other important concepts. But, in the subsequent chapters we concentrate on how to construct push down automaton and how it can be constructed to accept context free language.

5.1 Advantages of regular and non-regular languages

The language obtained from regular grammar is called regular language. Using regular languages we can describe some patterns very effectively such as recognition of identifiers, numbers, identification of comments in a program etc. But, the non-regular languages cannot be described using regular grammar which is the limitation of the grammar.

The non-regular languages are very important in programming languages so as to identify the matching parentheses, to match the nested if statements, whether a statement is syntactically correct or not and so on. So, apart from regular languages, non-regular languages also play a very important role in programming languages and the language translators such as interpreters and compilers. So, in this chapter let us discuss and study the way to represent a context free grammar using which we obtain some of the non-regular languages.

5.2 Context free grammars

In the regular grammar there is restriction on the number of non-terminals on the left hand side as well as on the right hand side. On the left hand side of the production, there will be only one symbol and that symbol must be a non-terminal (variable). But, the right hand side of the production may contain zero or more terminals. If a non-terminal (variable) is present, that non-terminal should be the left most symbol or the right most symbol. But, in a context free grammar there is only one restriction. The symbol on the left hand side of the production should be a single non-terminal but, there is no restriction on the right hand side of the production. Any number of non-terminals and terminals may be present (including ε). Formally, the context free grammar is defined as follows.

Definition: A grammar G is a quadruple (having four components) G = (V, T, P, S) where

- V is set of variables or non-terminals
- T is set of terminals
- P is set of productions
- S is the start symbol

is said to be type 2 grammar or context free grammar (CFG) if all the productions are of the form

$$A \rightarrow \alpha$$

where $\alpha \in (VUT)^*$ and $A \in V$. The symbol ε (indicating NULL string) can appear on the right hand side of any production.

Note: Some of the observations made in this definition are:

1. There is only one symbol A on the left hand side of the production and that symbol must be a non-terminal
2. $\alpha \in (VUT)^*$ implies that right hand side of the string may contain any number of terminals and non-terminals including ε (NULL string).

The language generated from this grammar is called type 2 language or context free language (CFL). For example, the following grammar is a context free grammar:

$$
\begin{aligned}
S &\rightarrow aBaa \mid bA \mid \varepsilon \\
A &\rightarrow aA \mid bAA \\
B &\rightarrow BbB \mid a \mid \varepsilon
\end{aligned}
$$

Every regular grammar is a CFG and hence a regular language is also context free language but the reverse is not true always. Basically, regular grammar is subset of context free grammar and regular language is subset of context free language. Some of the context free languages are discussed below.

Example 5.1: Let G = (V,T,P,S) be a CFG where

V = { S }
T = {a, b}
P = {
S → aSa | bSb | ε
}
S is the start symbol.
What is the language generated by this grammar?

The null string ε can be obtained by applying the production S → ε and the derivation is shown below:

$S \Rightarrow \varepsilon$ (By applying $S \rightarrow \varepsilon$)

Consider the derivation

$S \Rightarrow aSa$ (By applying $S \rightarrow aSa$)
$\Rightarrow abSba$ (By applying $S \rightarrow bSb$)
$\Rightarrow abbSbba$ (By applying $S \rightarrow bSb$)
$\Rightarrow abbbSbbba$ (By applying $S \rightarrow bSb$)
$\Rightarrow abbbbbba$ (By applying $S \rightarrow \varepsilon$)

So, by applying the productions

$$S \rightarrow aSa \text{ and } S \rightarrow bSb$$

any number of times and in any order and finally applying the production

$$S \rightarrow \varepsilon$$

we get a string *w* followed by reverse of *w* denoted by w^R. So, the language generated by this grammar is

$$L = \{w\, w^R \mid w \in \{a+b\}^*\}$$

As this language is generated from the context free grammar, this is context free language. We saw in chapter 4 that this language is not regular and FA can not be constructed for this.

Example 5.2: Show that the language L = { $a^m b^n$ | $m \neq n$ } is context free.

If it is possible to construct a CFG to generate the given language then we say that the language is context free. Let us construct the CFG for the language defined. It is clear from the given language that *m* number of a's are followed by *n* number of b's and number of a's and b's are not equal. As a first attempt we can have the production

$$S \rightarrow aSb$$

using which *n* number of a's are followed by one S followed by *n* number of b's are generated. Now, if we replace the non terminal S by the production

$$S \rightarrow \varepsilon$$

we get *n* number of a's followed by *n* number of b's. But, we should see that number of a's and b's are different. So, we should be in a position to generate either one or more extra a's or one or more extra b's. Hence, instead of the production

$$S \rightarrow \varepsilon$$

we can have the productions

$$S \rightarrow A \mid B$$

From the non-terminal A we can generate one or more a's and from non-terminal B we can generate one or more b's as shown below:

$$A \rightarrow aA \mid a$$
$$B \rightarrow bB \mid b$$

So, the context free grammar G = (V, T, P, S) where

V = { S, A, B}
T = {a, b}
P = {
S → aSb | A | B
A → aA | a
B → bB | b
}
S is the start symbol

OR

V = { S^1, S, A, B}
T = {a, b}
P = {
S^1 → AS | SB
S → aSb | ε
A → aA | a
B → bB | b
}
S^1 is the start symbol

generates the language L = { $a^m b^n$ | $m \neq n$ }. Note that both the grammars specified generate the same language L. Since a CFG exists for the language, the language is context free.

Example 5.3: Obtain a CFG to generate a language consisting of equal number of a's and b's.

Note: The solution is already provided in 1.34. It can also be solved using the following grammar.

The above language can also be represented as

$$L = \{w \mid w \in \{a, b\}^*, n_a(w) = n_b(w)\}$$

The context free grammar G = (V, T, P, S) where

V = {S, A, B}
T = {a, b}
P = {
S → aB | bA
A → aS | bAA | a
B → bS | aBB | b
}
S is the start symbol

generates the language consisting of equal number of a's and b's.

Example 5.4: Obtain a CFG on {a, b} to generate a language $L = \{a^n w w^R b^n \mid w \in \Sigma^*, n \geq 1\}$

This is similar to the problem shown in example 5.1 which can generate the string ww^R. But, in this case the string ww^R must be enclosed between a^n and b^n where $n \geq 1$. The final grammar is

V = { S }
T = {a, b}
P = {
S → aSb | aAb
A → aAa | bAb | ε
}
S is the start symbol.

It is left to the reader to verify whether the required language is generated by this or not. Show the derivation for the string which can generate the required language.

Example 5.5: Obtain a context free grammar to generate properly nested parentheses structures involving three kinds of parentheses ( ), [ ] and { }.

This is similar to the problem $a^n b^n$ where *a* is replaced by left parentheses and *b* is replaced by right parentheses, with little bit of modification. The complete grammar is shown below:

V = { S }
T = {(,), [,], {, }}
P = {
S → SS | (S) | [S] | {S} | ε
}
S is the start symbol.

It is left to the reader to verify whether the required language is generated by this or not. Show the derivation for the string which can generate the required language.

Example 5.6: Obtain a context free grammar to generate the following language
$$L = \{w \mid w \in \{a, b\}^*, n_a(v) \geq n_b(v) \text{ where } v \text{ is any prefix of } w\}$$

This is similar to the previous problem where '(' is replaced by *a* and ')' is replaced by *b* with little bit of modification. The complete grammar is shown below:

$V = \{ S \}$
$T = \{a, b\}$
$P = \{$
$\quad S \rightarrow SS \mid aSb \mid \varepsilon$
$\}$
S is the start symbol.

It is left to the reader to verify whether the required language is generated from the above grammar or not. Show the derivation for the string which can generate the required language.

Example 5.7: Obtain a context free grammar to generate the following language
$$L = \{01(1100)^n110(10)^n \mid n \geq 0\}$$

The complete grammar is shown below:

$V = \{ S \}$
$T = \{0, 1\}$
$P = \{$
$\quad S \rightarrow 01A$
$\quad A \rightarrow 1100A10 \mid 110$
$\}$
S is the start symbol.

It is left to the reader to verify whether the required language is generated by this or not. Show the derivation for the string which can generate the required language.

Example 5.8: Is the following language Context free?
$$L = \{a^nb^n \mid n \geq 0\}$$

If it is context free language, it should be generated by the context free grammar. Consider the context free grammar shown below:

$V = \{ S \}, T = \{a, b\}$

$P = \{$
$\quad S \rightarrow aSb \mid \varepsilon$
$\}$

and S is the start symbol. It is evident from the above grammar that the language generated by this grammar is $a^n b^n \mid n \geq 0$. The reader should show the derivation to generate a string $w \in L$. Since this grammar generates the language specified, the language is context free.

Example 5.9: Obtain a context free grammar to generate the following language

$$L = \{a^n b^m : m > n \text{ and } n \geq 0\}$$

It is similar to the previous problem, except that it has more number of *b*'s following *a*'s. The complete grammar is shown below:

$V = \{ S, B \}$
$T = \{a, b\}$
$P = \{$
$\quad S \rightarrow aSb \mid B$
$\quad B \rightarrow bB \mid b$
$\}$
S is the start symbol.

It is left to the reader to verify whether the required language is generated by this or not. Show the derivation for the string which can generate the required language.

Example 5.10: Obtain a CFG to generate unequal number of a's and b's

The above language can also be represented as

$$L = \{w \mid w \in \{a, b\}^*, n_a(w) \neq n_b(w)\}$$

Let us first construct a CFG G_1 which accepts a language consisting of more a's than b's. On similar lines we can construct a CFG G_2 which accepts a language consisting of more b's than a's. If A is the start symbol of G_1 and B is the start symbol of G_2, then these two grammars can be combined into a single CFG G which accepts both the languages using the production

$$S \rightarrow A \mid B \qquad (5.1)$$

First we shall see how to obtain G1 which accepts more a's than b's. The language can be recursively defined as follows:

1. The symbol 'a' has more a's than b's
2. If X has more number of a's than b's then Xa and aX definitely contain more a's than b's.

The production corresponding to these two definitions can be written as

$$A \rightarrow a \mid aA \qquad (5.2)$$

The above production contains only a's and do not contain any b's. We should add b's so that number of a's remain more than number of b's. This can be achieved by adding symbol 'b' to the

left or middle or to right of two elements X and Y ∈ L_1 which is the language corresponding to the grammar G_1. The resulting productions using this definition are

$$A \rightarrow bAA \mid AbA \mid AAb \qquad (5.3)$$

So, the grammar G_1 to produce more a's than b's can be obtained by combining the productions obtained in (5.2) and (5.3) as shown below:

$$A \rightarrow a \mid Aa \mid aA \mid bAA \mid AbA \mid AAb \qquad (5.4)$$

On similar lines, we can write a grammar G_2 to produce more b's than a's as shown below:

$$B \rightarrow b \mid bB \mid aBB \mid BaB \mid BBa \qquad (5.5)$$

So, the final grammar G to generate unequal number of a's and b's can be obtained by combining the productions obtained in (5.1), (5.4) and (5.5) as shown below. The grammar G = (V, T, P, S) where

V = {S, A, B}
T = {a, b}
P = {
S → A | B
A → a | aA | bAA | AbA | AAb
B → b | bB | aBB | BaB | BBa
}
S is the start symbol

produces unequal number of a's and b's

Example 5.11: For the regular expression (011+1)*(01)* obtain the context free grammar.

The regular expression (011 + 1)*(01)* is of the form A*B* where A can be 011 or 1 and B is 01. The regular expression A*B* means that any number of A's (possibly none) are followed by any number of B's(possibly none). Any number of A's (i.e, 011's or 1's) can be generated using the productions

$$A \rightarrow 011A \mid 1A \mid \varepsilon$$

Any number of B's (i.e., 01's) can be generated using the productions

$$B \rightarrow 01B \mid \varepsilon$$

Now, the language generated from the regular expression (011 + 1)*(01)* can be obtained by concatenating A and B using the production

$$S \rightarrow AB$$

So, the final grammar is G = (V, T, P, S) where

V = {S, A, B}
T = {0, 1}
P = {
S → AB
A → 011A | 1A | ε
B → 01B | ε
}
S is the start symbol

Example 5.12: Obtain a grammar to generate an arithmetic expression using the operators +, -, *, / and ^(indicating power). An identifier can start with any of the letters from {a, b, c} and can be followed by zero or more symbols from {a, b, c}

An arithmetic expression can be recursively defined as follows:

1. An expression E can be an identifier
2. If E is any arithmetic expression then
   i. E + E
   ii. E - E
   iii. E * E
   iv. E / E
   v. E ^ E
   vi. (E)

   are all arithmetic expressions.

An identifier I of length at least one can be generated using any combinations of a's, b's and c's using the following productions

$$I \rightarrow Ia \mid Ib \mid Ic \mid a \mid b \mid c$$

By first definition of an arithmetic expression I is an arithmetic expression which can be obtained using the production

$$E \rightarrow I$$

By the second definition of an arithmetic expression the various productions that can be obtained are:

$$E \rightarrow E+E$$
$$E \rightarrow E-E$$
$$E \rightarrow E*E$$
$$E \rightarrow E/E$$
$$E \rightarrow E\hat{}E$$
$$E \rightarrow (E)$$

So, the complete grammar to generate an arithmetic expression is given by G = (V, T, P, S) where

V = {E,I}
T = {+, -, *, /, ^, a, b, c}
P = {
E → I
E → E+E
E → E-E
E → E*E
E → E/E
E → E^E
E → (E)
I → Ia | Ib | Ic | a | b | c
}
S is the start symbol

The productions in P can also be written as shown below:

E → I
E → E+E| E−E|E*E|E/E|E^E|(E)
I → Ia | Ib | Ic | a | b | c

5.3 Leftmost derivation

The section 1.16 gives the information with respect to *derivation.* The derivation process is shown in example 1.26. The reader is advised to refer this section before proceeding further. The derivation process is repeated for the sake of convenience in this section.

Example 5.13: Consider the grammar shown below from which any arithmetic expression can be obtained.

E → E+E
E → E-E
E → E*E
E → E/E
E → E^E
E → **id**

Note: The symbol ^ denotes exponentiation.

The non-terminal E is used instead of using the word *expression.* The left-hand side of the first production i.e., E is considered to be the start symbol. Obtain the string id + id * id and show the derivation for the same.

Solution: The derivation to get the string id + id * id is shown below.

$$E \underset{lm}{\Rightarrow} E+E$$
$$\Rightarrow \textbf{id}+E$$
$$\Rightarrow \textbf{id}+E*E$$
$$\Rightarrow \textbf{id}+\textbf{id}*E$$
$$\Rightarrow \textbf{id}+\textbf{id}*\textbf{id}$$

Note that at each step in the derivation process, only the left most variable E is replaced and so the derivation is said to be leftmost. The string **id + id * id** is obtained from the start symbol E by applying leftmost derivation and can be written as

$$E \overset{+}{\underset{lm}{\Rightarrow}} \mathbf{id + id * id}$$

indicating the string **id + id * id** is derived from E by applying more than one production. The symbol $\overset{+}{\underset{lm}{\Rightarrow}}$ denotes that one or more steps are used in the derivation where as the symbol $\overset{*}{\underset{lm}{\Rightarrow}}$ denotes that zero or more steps are used in the derivation. Thus, the leftmost derivation can be defined as follows:

Definition: In the derivation process if a left most variable is replaced at every step, then the derivation is said to be leftmost and is shown in example 5.13. It is clear from leftmost derivation that each of the following:

$$\{E, E + E, \mathbf{id} + E, \mathbf{id} + E * E, \mathbf{id} + \mathbf{id} * E, \mathbf{id} + \mathbf{id} * \mathbf{id}\}$$

can be obtained from the start symbol. Each string in the set is called the sentential form. The formal definition of *sentential form* is shown below:

Definition: Let G = (V, T, P, S) be a CFG. Any string $w \in (V \cup T)^*$ which is derivable from the start symbol S (denoted by $S \overset{*}{\Rightarrow} \alpha$) is called a sentence or *sentential form* of G. If there is a derivation of the form $S \overset{*}{\underset{lm}{\Rightarrow}} \alpha$, where at each step in the derivation process only a left most variable is replaced, then α is called *left-sentential* form and if there is a derivation of the form $S \overset{*}{\underset{rm}{\Rightarrow}} \alpha$, where at each step in the derivation process only a right most variable is replaced, then α is called *right-sentential* form.

Example 5.14: Obtain the leftmost derivation for the string aaabbabbba using the following grammar.

$$\begin{aligned} S &\rightarrow aB \mid bA \\ A &\rightarrow aS \mid bAA \mid a \\ B &\rightarrow bS \mid aBB \mid b \end{aligned}$$

The leftmost derivation for the string aaabbabbba is shown below:

$$\begin{aligned} S &\overset{lm}{\Rightarrow} aB && \text{(Applying } S \rightarrow aB) \\ &\Rightarrow aaBB && \text{(Applying } B \rightarrow aBB) \\ &\Rightarrow aaaBBB && \text{(Applying } B \rightarrow aBB) \\ &\Rightarrow aaabBB && \text{(Applying } B \rightarrow b) \\ &\Rightarrow aaabbB && \text{(Applying } B \rightarrow b) \\ &\Rightarrow aaabbaBB && \text{(Applying } B \rightarrow aBB) \\ &\Rightarrow aaabbabB && \text{(Applying } B \rightarrow b) \\ &\Rightarrow aaabbabbS && \text{(Applying } B \rightarrow bS) \\ &\Rightarrow aaabbabbbA && \text{(Applying } S \rightarrow bA) \\ &\Rightarrow aaabbabbba && \text{(Applying } A \rightarrow a) \end{aligned}$$

5.4 Rightmost derivation

Definition: In the derivation process if a right most variable is replaced at every step, then the derivation is said to be rightmost.

The rightmost derivation for the grammar shown in example 5.13 is shown below.

$$\begin{aligned} E &\underset{rm}{\Rightarrow} E + E \\ &\Rightarrow E + E * E \\ &\Rightarrow E + E * \mathbf{id} \\ &\Rightarrow E + \mathbf{id} * \mathbf{id} \\ &\Rightarrow \mathbf{id} + \mathbf{id} * \mathbf{id} \end{aligned}$$

Note that at each step in the derivation process, only the right most variable E is replaced and so the derivation is said to be right most. The string **id + id * id** is obtained from the start symbol E by applying right most derivation and can be written as

$$E \overset{+}{\underset{rm}{\Rightarrow}} \mathbf{id} + \mathbf{id} * \mathbf{id}$$

5.5 Derivation Tree (Parse tree)

The derivation can be shown in the form of a tree. Such trees are called derivation or parse trees. The leftmost derivation as well as the right most derivation can be represented using derivation trees. The derivation tree can be defined as shown below.

Definition (Parse tree or Derivation Tree): Let G = (V, T, P, S) be a CFG. The tree is derivation tree (parse tree) with the following properties.

1. The root has the label S.
2. Every vertex has a label which is in (V U T U ε).
3. Every leaf node has label from T and an interior vertex has a label from V.
4. If a vertex is labeled A and if X_1, X_2, X_3, ... X_n are all children of A from left, then $A \rightarrow X_1X_2X_3....X_n$ must be a production in P.

The parse tree for the right most derivation shown in section 5.5 is shown in figure 5.1

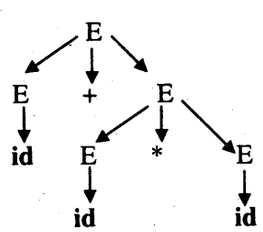


Fig.5.1 Derivation tree

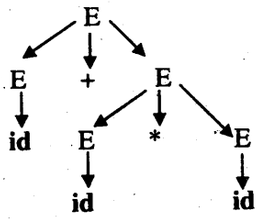
In the above derivation tree, let us read only the leaf nodes from left to right (leaving the ε-symbols if any). The string obtained is

id +id*id

is called the *yield* of the tree. Thus, the *yield* of a tree can be formally defined as follows:

Definition: The *yield* of a tree is the string of symbols obtained by only reading the leaves of the tree from *left* to *right* without considering the ε-symbols. The yield of the tree is derived always from the root and the yield of the tree is always a terminal string.

For example, consider the derivation tree (or parse tree) shown below:



If we read only the terminal symbols in this tree from left to right we get **id + id * id** and is the yield of the given parse tree. The partial parse tree(partial derivation tree) is defined as follows.

Definition (Partial Parse tree or Partial Derivation Tree):
Let G = (V, T, P, S) be a CFG. The tree is partial derivation tree (parse tree) with the following properties.

1. The root has the label S.
2. Every vertex has a label which is in $(V \cup T \cup \varepsilon)$.
3. Every leaf node has label from $(V \cup T \cup \varepsilon)$.
4. If a vertex is labeled A and if $X_1, X_2, X_3, \ldots X_n$ are all children of A from left, then $A \rightarrow X_1X_2X_3 \ldots X_n$ must be a production in P.

Note: The difference between parse tree and partial parse tree is same except in the property 3 mentioned.

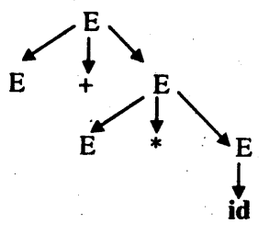
Consider the partial derivation (by applying right most derivation) for the grammar

$$E \rightarrow E + E \mid E * E \mid id$$

is shown below

$$\begin{aligned} E &\Rightarrow E + E \\ &\Rightarrow E + E * E \\ &\Rightarrow E + E * \mathbf{id} \end{aligned}$$

For this partial right most derivation, the partial derivation tree is shown below:



It is clear from the *parse* tree and *partial parse* tree that all the leaves in parse tree are the symbols from (T $\cup$ ε) whereas in partial parse tree the leaves will be from (V $\cup$ T $\cup$ ε).

5.6 Ambiguous grammar

Definition: Let G = (V, T, P, S) be a context free grammar. A grammar G is *ambiguous* if and only if there exists at least one string $w \in T^*$ for which two or more different parse trees exist by applying either the left most derivation or right most derivation. Note that after applying leftmost derivation or right most derivation even though the derivations look different and if the structure of parse trees obtained are same, we can not jump to the conclusion that the grammar is ambiguous. It is not the multiplicity of the derivations that cause ambiguity. But, it is the existence of two or more parse trees for the same string w derived from the root labeled S.

Note:

1. Obtain the leftmost derivation and get a string w. Obtain the right most derivation and get a string w. For both the derivations construct the parse tree. If there are two different parse trees, then the grammar is ambiguous.
2. Obtain the string w by applying leftmost derivation twice and construct the parse tree. If the two parse trees are different, the grammar is ambiguous.
3. Obtain the string w by applying rightmost derivation twice and construct the parse tree. If the two parse trees are different, the grammar is ambiguous.
4. Apply the leftmost derivation and a get string. Apply the leftmost derivation again and a get a different string. The parse trees obtained will naturally be different and do not come to the conclusion that the grammar is ambiguous.

Example 5.15: Consider the grammar shown below from which any arithmetic expression can be obtained.

$$
\begin{aligned}
E &\rightarrow E + E \\
E &\rightarrow E - E \\
E &\rightarrow E * E \\
E &\rightarrow E / E \\
E &\rightarrow (E) \mid I \\
I &\rightarrow \mathbf{id}
\end{aligned}
$$

Show that the grammar is ambiguous.

Note: Leftmost derivation, rightmost derivation and parse trees are equivalent.

The sentence **id + id * id** can be obtained from leftmost derivation in two ways as shown below.

$$
\begin{aligned}
E &\Rightarrow E+E \\
&\Rightarrow \mathbf{id}+E \\
&\Rightarrow \mathbf{id}+E*E \\
&\Rightarrow \mathbf{id}+\mathbf{id}*E \\
&\Rightarrow \mathbf{id}+\mathbf{id}*\mathbf{id}
\end{aligned}
\qquad
\begin{aligned}
E &\Rightarrow E*E \\
&\Rightarrow E+E*E \\
&\Rightarrow \mathbf{id}+E*E \\
&\Rightarrow \mathbf{id}+\mathbf{id}*E \\
&\Rightarrow \mathbf{id}+\mathbf{id}*\mathbf{id}
\end{aligned}
$$

The corresponding derivation trees for the two leftmost derivations are shown in figure 5.2.

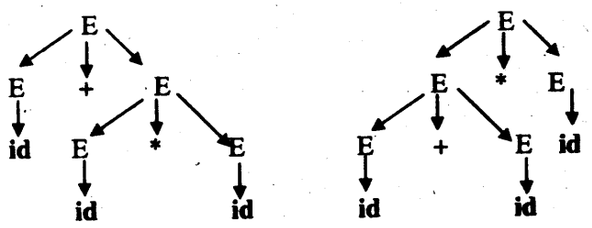


fig.5.2 Derivation trees for Leftmost derivation

Since the two parse trees are different for the same sentence **id + id * id** by applying leftmost derivation, the grammar is ambiguous.

Example 5.16: Is the following grammar ambiguous?

$$
\begin{aligned}
S &\rightarrow aS \mid X \\
X &\rightarrow aX \mid a
\end{aligned}
$$

Consider the two leftmost derivations for the string aaaa.

$$
\begin{aligned}
S &\Rightarrow aS \\
&\Rightarrow aaS \\
&\Rightarrow aaaS \\
&\Rightarrow aaaX \\
&\Rightarrow aaaa
\end{aligned}
\qquad
\begin{aligned}
S &\Rightarrow X \\
&\Rightarrow aX \\
&\Rightarrow aaX \\
&\Rightarrow aaaX \\
&\Rightarrow aaaa
\end{aligned}
$$

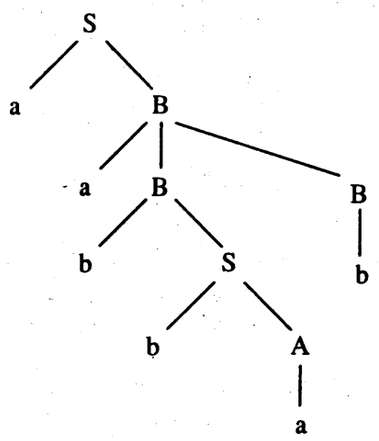
Since there are two leftmost derivations for the same sentence aaaa, the given grammar is ambiguous.

Example 5.17: Is the following grammar ambiguous?

$$
\begin{aligned}
S &\rightarrow aB \mid bA \\
A &\rightarrow aS \mid bAA \mid a \\
B &\rightarrow bS \mid aBB \mid b
\end{aligned}
$$

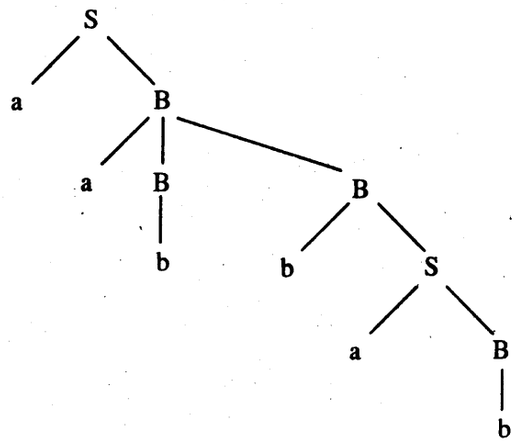
Solution: Consider the leftmost derivation and the corresponding pares tree shown below:

$S \Rightarrow aB$ (Applying $S \rightarrow aB$)
$\Rightarrow aaBB$ (Applying $B \rightarrow aBB$)
$\Rightarrow aabSB$ (Applying $B \rightarrow bS$)
$\Rightarrow aabbAB$ (Applying $S \rightarrow bA$)
$\Rightarrow aabbaB$ (Applying $A \rightarrow a$)
$\Rightarrow aabbab$ (Applying $B \rightarrow b$)



The same string aabbab can be obtained again by applying leftmost derivation as shown below:

$S \Rightarrow aB$ (Applying $S \rightarrow aB$)
$\Rightarrow aaBB$ (Applying $B \rightarrow aBB$)
$\Rightarrow aabB$ (Applying $B \rightarrow b$)
$\Rightarrow aabbS$ (Applying $B \rightarrow bS$)
$\Rightarrow aabbaB$ (Applying $S \rightarrow aB$)
$\Rightarrow aabbab$ (Applying $B \rightarrow b$)



Note that there are two parse trees for the string *aabbab* by applying leftmost derivation and so the given grammar is ambiguous.

Example 5.18: Obtain the string *aaabbabbba* by applying left most derivation and the parse tree for the grammar shown below. Is it possible to obtain the same string again by applying leftmost derivation but by selecting different productions?

$$\begin{aligned} S &\rightarrow aB \mid bA \\ A &\rightarrow aS \mid bAA \mid a \\ B &\rightarrow bS \mid aBB \mid b \end{aligned}$$

The leftmost derivation for the string aaabbabbba is shown below:

$$\begin{aligned} S &\Rightarrow aB && \text{(Applying } S \rightarrow aB) \\ &\Rightarrow aaBB && \text{(Applying } B \rightarrow aBB) \\ &\Rightarrow aaaBBB && \text{(Applying } B \rightarrow aBB) \\ &\Rightarrow aaabBB && \text{(Applying } B \rightarrow b) \\ &\Rightarrow aaabbB && \text{(Applying } B \rightarrow b) \\ &\Rightarrow aaabbaBB && \text{(Applying } B \rightarrow aBB) \\ &\Rightarrow aaabbabB && \text{(Applying } B \rightarrow b) \\ &\Rightarrow aaabbabbS && \text{(Applying } B \rightarrow bS) \\ &\Rightarrow aaabbabbbA && \text{(Applying } S \rightarrow bA) \\ &\Rightarrow aaabbabbba && \text{(Applying } A \rightarrow a) \end{aligned}$$

The parse tree for the above derivation is shown in figure 5.3.

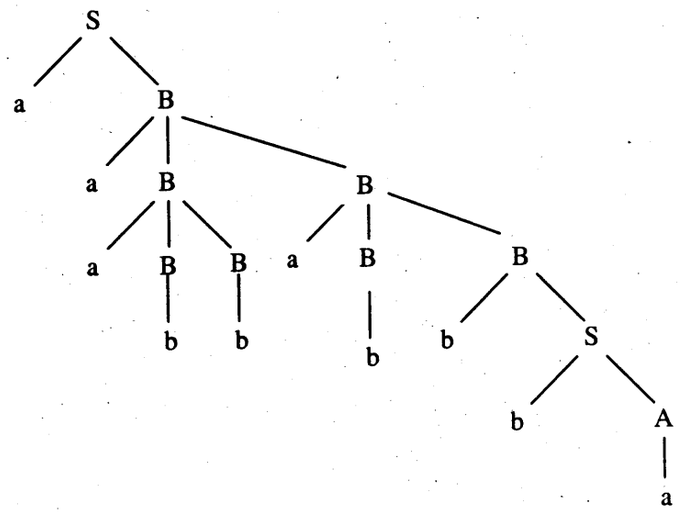


fig.5.3 Derivation tree for Leftmost derivation

The leftmost derivation for the same string aaabbabbba but by applying different set of productions is shown below:

S	$\Rightarrow$ aB	(Applying S $\rightarrow$ aB)
	$\Rightarrow$ aaBB	(Applying B $\rightarrow$ aBB)
	$\Rightarrow$ aaaBBB	(Applying B $\rightarrow$ aBB)
	$\Rightarrow$ aaabSBB	(Applying B $\rightarrow$ bS)
	$\Rightarrow$ aaabbABB	(Applying S $\rightarrow$ bA)
	$\Rightarrow$ aaabbaBB	(Applying A $\rightarrow$ a)
	$\Rightarrow$ aaabbabB	(Applying B $\rightarrow$ b)
	$\Rightarrow$ aaabbabbS	(Applying B $\rightarrow$ bS)
	$\Rightarrow$ aaabbabbbA	(Applying S $\rightarrow$ bA)
	$\Rightarrow$ aaabbabbba	(Applying A$\rightarrow$ a)

The parse tree for this derivation is shown in figure 5.4.

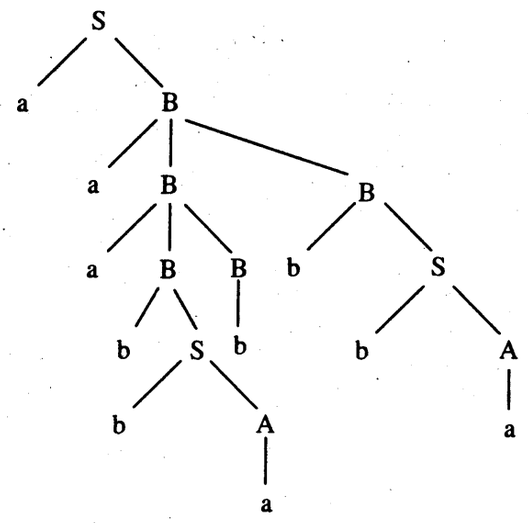


fig.5.4 Derivation tree for Leftmost derivation

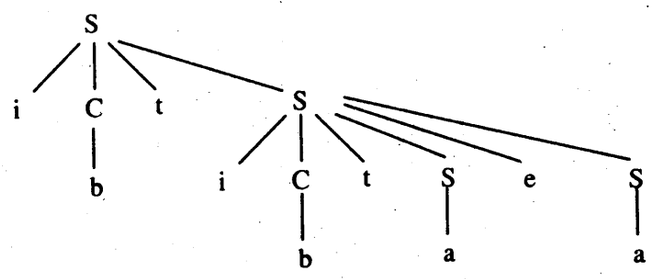
Example 5.19: Is the following grammar ambiguous?

S $\rightarrow$ iCtS | iCtSeS | a
C $\rightarrow$ b

The string ibtibtaea can be obtained by applying the leftmost derivation as shown below:

$$
\begin{aligned}
S &\Rightarrow iCtS \\
&\Rightarrow ibtS \\
&\Rightarrow ibtiCtSeS \\
&\Rightarrow ibtibtSeS \\
&\Rightarrow ibtibtaeS \\
&\Rightarrow ibtibtaea
\end{aligned}
$$

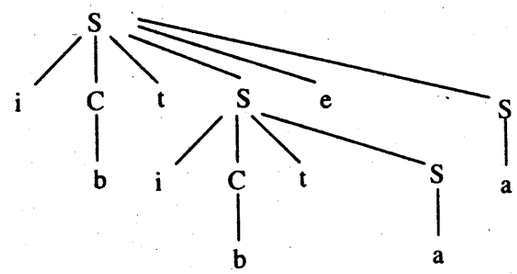
The parse tree for this is shown below:



The string ibtibtaea can be obtained again by applying the leftmost derivation but using different sets of productions as shown below:

$$
\begin{aligned}
S &\Rightarrow iCtSeS \\
&\Rightarrow ibtSeS \\
&\Rightarrow ibtiCtSeS \\
&\Rightarrow ibtibtSeS \\
&\Rightarrow ibtibtaeS \\
&\Rightarrow ibtibtaea
\end{aligned}
$$

The parse tree for this is shown below:



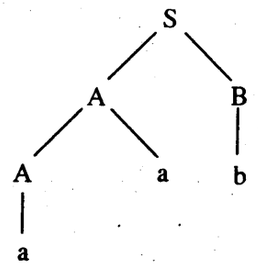
Since there are two different parse trees for the string 'ibtibtaea' by applying leftmost derivation the given grammar is ambiguous.

Example 5.20: Is the grammar ambiguous?

$$S \rightarrow AB \mid aaB$$
$$A \rightarrow a \mid Aa$$
$$B \rightarrow b$$

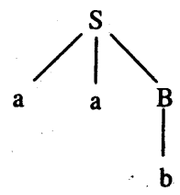
Consider the left most derivation for the string *aab* and the corresponding pares tree

$$S \Rightarrow AB \Rightarrow AaB \Rightarrow aaB \Rightarrow aab$$



Consider the left most derivation again for the string *aab* but using different set of productions along with the parse tree

$$S \Rightarrow aaB \Rightarrow aab$$



Since there are two parse trees for the string *aab*, the given grammar is ambiguous.

Example 5.21: Show that the following grammar is ambiguous

$$S \rightarrow aSbS$$
$$S \rightarrow bSaS$$
$$S \rightarrow \varepsilon$$

Consider the leftmost derivation for the string *aababb* and the corresponding parse tree

$$S \Rightarrow aSbS \quad \text{by using } S \rightarrow aSbS$$
$$\Rightarrow aaSbSbS \quad \text{by using } S \rightarrow aSbS$$
$$\Rightarrow aabSaSbSbS \quad \text{by using } S \rightarrow bSaS$$
$$\Rightarrow aabaSbSbS \quad \text{by using } S \rightarrow \varepsilon$$
$$\Rightarrow aababSbS \quad \text{by using } S \rightarrow \varepsilon$$
$$\Rightarrow aababbS \quad \text{by using } S \rightarrow \varepsilon$$
$$\Rightarrow aababb \quad \text{by using } S \rightarrow \varepsilon$$