Chapter 4

Context Free Grammars & Languages

What are we studying in this chapter?

- Context free grammars
- Parse trees
- Applications
- Ambiguity in grammars and Languages. - 6 hours

4.1 Grammar

In this section, let us see the definition of a grammar and how a language can be constructed using the grammar and various types of grammars. Consider the sentence:

Monalika ate slowly

Noun verb adverb

Thus, a sentence in this example starts with *noun* followed by *verb* followed by *adverb*. After replacing *noun*, *verb* and *adverb* with appropriate words, a grammatically correct sentence can be obtained. The rules to form this sentence can be written as

1. sentence → <noun> <verb> <adverb>
2. noun → Monalika
3. verb → ate
4. adverb → slowly

Now, let us see "What are the various components of the above grammar?" The various components of the grammar are shown below:

Components of the grammar
- Variables
- Terminals
- Productions
- Start symbol

Scanned with CamScanner

Variables: In the above four rules *sentence*, *noun*, *verb*, and *adverb* are called *variables*. The variables are also called *non-terminals*.

Terminals: The words "Monalika", "ate" and "slowly" are called terminals.

Productions: The above four rules which are used to obtain the sentence "Monalika ate slowly" are called productions.

- Each production starts with *non-terminal*
- Followed by an *arrow*
- Followed by combinations of *non-terminals* and/or *terminals*.

Start symbol: The left hand side of '→' in the first production is called the *start symbol*. The start symbol is usually denoted by the letter S

Note: The *non-terminals* can be replaced by terminals or non-terminals whereas the *terminals* cannot be replaced.

Now, let us see "What is a grammar? Give an example."

Definition: A grammar G is 4-tuple or quadruple G = (V, T, P, S) where

- V is set of variables or non-terminals.
- T is set of terminals
- P is set of productions. Each production is of the form $\alpha \rightarrow \beta$ where α is a string in $(V \cup T)^+$ and hence α can not be ε and ε can not occur on the right hand side of any production. But, β is a string in $(V \cup T)^*$ and hence it includes ε also. So, ε can occur on the right hand side of the production.
- S is the start symbol

Note: The null string is denoted by symbol ε(epsilon). The null string cannot occur on the left hand side of any production

Before constructing grammar, let us see "What are the notations used while constructing the grammar?" The various notations used are shown below:

- The following are the terminals
  - The keywords such as if, for, while, do-while etc.
  - Digits from 0 to 9
  - Symbols such as +, -, *, / etc

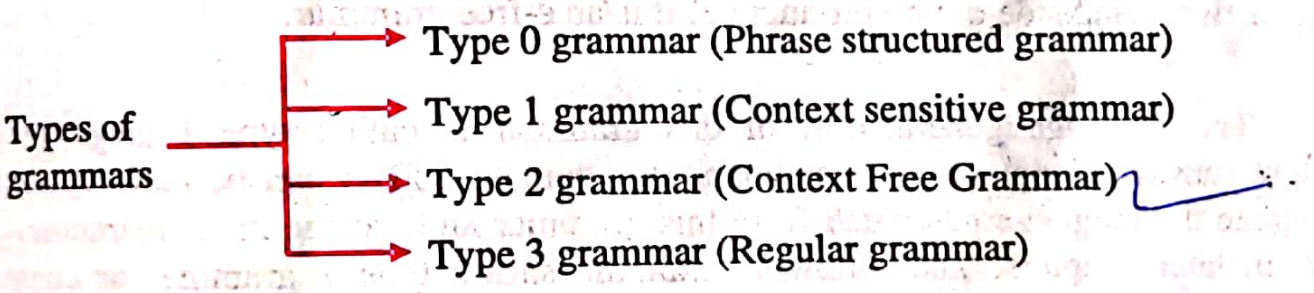
Scanned with CamScanner

- The lower case letters near the beginning of the alphabets such as *a*, *b*, *c*, *d* etc.
- The bold faced letters such as **id**

- The following are non-terminals
  - The lower case names such as expression, operator, operand, statement etc
  - The capital letters near the beginning of the alphabets such as A, B, C, D etc
  - The letter S is the start symbol
- The lower case letters near the end of the alphabets such as u, v, w, x, y, z represent string of terminals.
- The capital letters near the end of the alphabets such as X, Y, Z etc represent grammar symbols. The grammar symbol can be terminal or non-terminal.
- The Greek letters such as represent string of grammar symbols

4.2 Chomsky Hierarchy

Noam Chomsky who is the founder of formal language theory has classified the grammar into various categories. Now, let us see "What are various types of grammars?" or "Explain Chomsky Hierarchy?" or "Explain the classification of grammars" The grammar can be classified as shown below:



Scanned with CamScanner

4.2.3 Type 2 grammar or context free grammar

Now, let us see "What is Type 2 grammar or context free grammar?" The type 2 grammar also called context free grammar (CFG) is defined as follows.

Definition: A grammar G = (V, T, P, S) is said to be type 2 grammar or context free grammar if all the productions are of the form $A \rightarrow \alpha$ where $\alpha \in (VUT)^*$ and A is non-terminal. The symbol ε can appear on the right hand side of any production. The

Scanned with CamScanner

context free grammar was discussed in section 1.14. The language generated from this grammar is called type 2 language or context free language. Pushdown automaton (PDA) can be constructed to recognize the language generated from this grammar. The grammar

$$\begin{aligned} S &\rightarrow aB \mid bA \mid \varepsilon \\ A &\rightarrow aA \mid b \\ B &\rightarrow bB \mid a \mid \varepsilon \end{aligned}$$

is an example for context free grammar.

Scanned with CamScanner

4.3 Grammar from Finite Automata

Now, let us see "What is the easiest method of constructing a grammar?" A grammar can be obtained very easily using finite automaton. The general procedure is shown below:

Procedure: Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automata accepting L where

$Q = \{q_0, q_1, \ldots q_n\}$

$\Sigma = \{a_1, a_2, \ldots a_m\}$

Scanned with CamScanner

A grammar G = (V, T, P, S) can be constructed where

- $V = \{q_0, q_1, \ldots q_n\}$ i.e., the states of DFA will be the variables in the grammar.
- $T = \Sigma$ i.e., the input alphabets of DFA are the terminals in grammar.
- $S = q_0$ i.e., the start state of DFA is the start symbol in the grammar.
- The productions P from the transitions can be obtained as shown below:

Step 1: If $\delta(q_i, a) = q_j$ then introduce the transition:

$$q_i \rightarrow aq_j$$

Step 2: If $q \in F$ i.e., if q is the final state in FA, then introduce the production

$$q \rightarrow \varepsilon$$

Using the above two steps we can convert any finite automaton (DFA or NFA or ε-NFA) into grammar.

Now, let us see "How to obtain context free grammar for simple languages using finite automata?" The table shows the DFA accepting the language, the transitions defined and the equivalent grammar:

Example 1: Obtain grammar to generate string consisting of any number of a's

Solution: The DFA to accept string consisting of any number of a's is shown below:

DFA	Transitions	Grammar
	• S is a final state	$S \rightarrow \varepsilon$
	• $\delta(S, a) = S$	
	$S \rightarrow a \quad S$	$S \rightarrow aS$

So, the grammar to generate string consisting of any number of a's is given by:

$$S \rightarrow \varepsilon$$
$$S \rightarrow aS$$

Grammar

The language generated by grammar can be formally written as shown below:

$$L = \{a^n : n \geq 0\}$$

Scanned with CamScanner

Example 2: Obtain grammar to generate string consisting of at least one *a*.

Solution: The DFA to accept string consisting of at least one *a* is shown below:

Method 1: By replacing ε by *a* in the above grammar, the minimum string generated is *a*. So, the grammar to generate string consisting of at least one *a* is shown below:

$$\left.\begin{array}{l} S \rightarrow a \\ S \rightarrow aS \end{array}\right\} \text{ Grammar to generate at least one } a$$

Note: The above productions start from S and hence can also be written as:

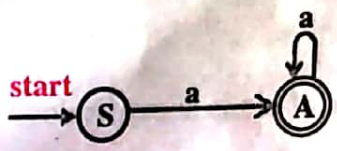
$$S \rightarrow a \mid aS$$

The language generated by grammar can be formally written as shown below:

$$L = \{a^n : n \geq 1\}$$

Method 2: The above grammar can be generated by constructing a DFA and then converting DFA into equivalent grammar as shown below:

DFA



Transitions

Grammar

- $\delta(S, a) = A$ ⇒ $S \rightarrow aA$

  $S \rightarrow a \quad A$

- $\delta(A, a) = A$ ⇒ $A \rightarrow aA$

  $A \rightarrow a \quad A$

- A is a final state $\quad A \rightarrow \varepsilon$

So, the final grammar to generate to at least one *a* is shown below:

$$\left.\begin{array}{l} S \rightarrow aA \\ A \rightarrow aA \\ A \rightarrow \varepsilon \end{array}\right\} \text{ Grammar to generate at least one } a$$

The language generated by grammar can be formally written as shown below:

$$L = \{a^n : n \geq 1\}$$

Note: Observe from method 1 and method 2 that one or more different grammars may exist that generate same language.

Scanned with CamScanner

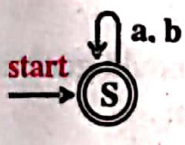
Note: For each final state introduce the ε transition. For example, in a DFA, if the states A and B are final states, then in the grammar we should introduce two ε-transitions:

$A \rightarrow \varepsilon$
$B \rightarrow \varepsilon$

Example 3: Obtain grammar to generate string consisting of any number of a's and b's

Solution: : The DFA to accept string consisting of any number of a's and b's is shown below:

DFA



Transitions

- S is a final state — **Grammar** $S \rightarrow \varepsilon$
- $\delta(S, a) = S$
  $S \rightarrow a \quad S$ ⇨ $S \rightarrow aS$
- $\delta(S, b) = S$
  $S \rightarrow b \quad S$ ⇨ $S \rightarrow bS$

So, the grammar to generate string consisting of any number of a's and b's is given by:

$S \rightarrow \varepsilon$
$S \rightarrow aS$
$S \rightarrow bS$

} **Grammar to generate string consisting of any number of a's and b's**

Note: Since all productions start from S, the above productions can also be written as:

$S \rightarrow \varepsilon \mid aS \mid bS$

The language generated by grammar can be formally written as shown below:

$$L = \{(a+b)^n : n \geq 0\}$$

Example 4: Obtain grammar to generate string consisting of at least two a's.

Solution: In the example 2 (page 4.8, method 1) we have a production

$S \rightarrow a$

Scanned with CamScanner

using which one *a* is generated. To generate at least two a's, in the above production replace one *a* by two *a*'s. So, the grammar to generate string consiting of at least two a's is given by:

$$S \rightarrow aa$$
$$S \rightarrow aS$$

Grammar

Example 5: Obtain grammar to generate string consisting of even number of a's.

Solution: The given language can be written as:

$$L=\{\varepsilon, aa, aaaa, aaaaaa, \ldots\ldots\}$$

Or

$$L = \{w : n_a(w) \bmod 2 = 0 \text{ where } w \in a^*\}$$

In the example 1 (page 4.7) we have productions

$$S \rightarrow \varepsilon \mid aS$$

To get the required language we should have ε, but subsequently we should generate multiples of two a's. This is done by replacing *a* by *aa* in the production:

$$S \rightarrow aS$$

So, the grammar is given by:

$$S \rightarrow \varepsilon \mid aaS$$

Grammar

Example 6: Obtain grammar to generate string consisting of multiples of three a's.

Solution: The given language can be written as:

$$L=\{\varepsilon, aaa, aaaaaa, aaaaaaaaa, \ldots\ldots\}$$

or

$$L = \{w : n_a(w) \bmod 3 = 0 \text{ where } w \in a^*\}$$

In the previous problem instead of aa we replace it by aaa. So, the final grammar is:

$$S \rightarrow \varepsilon \mid aaaS$$

Grammar

Example 7: Obtain grammar to generate strings of a's and b's such that string length is multiple of 3.

Solution: The grammar shown in example 6 can also be written as:

$$S \rightarrow \varepsilon \mid AAAS$$
$$A \rightarrow a$$

Scanned with CamScanner

By replacing *a* by *a* or *b* we get strings of a's and b's whose length is a multiple of three. So, the grammar is:

$S \rightarrow \varepsilon \mid AAAS$

$A \rightarrow a \mid b$

Grammar

Example 8: Obtain grammar to generate string consisting of any number of a's and b's with at least one *a*.

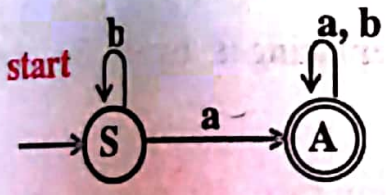
Solution: (Method 1): Same as grammar shown in example 3 but replacing ε by *a*. So, the resultant grammar is:

$S \rightarrow a$
$S \rightarrow aS$
$S \rightarrow bS$

} **Grammar to generate strings of a's and b's with at least one a**

Method 2: The DFA to accept strings of a's and b's consisting of at least one *a* is shown below:

DFA

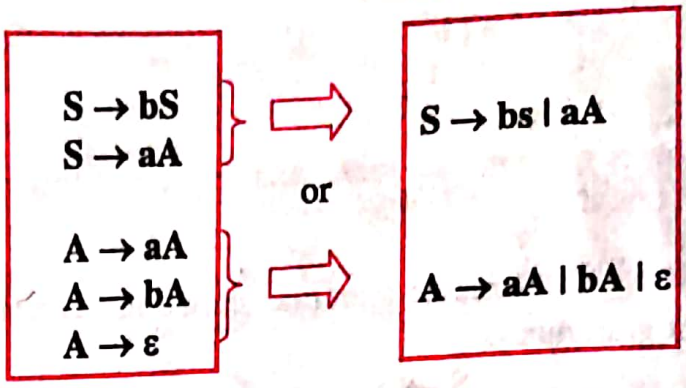


Transitions **Grammar**

- $\delta(S, b) = S$
  $S \rightarrow b \quad S$ ⇨ $S \rightarrow bS$
- $\delta(S, a) = A$
  $S \rightarrow a \quad A$ ⇨ $S \rightarrow aA$
- $\delta(A, a) = A$
  $A \rightarrow a \quad A$ ⇨ $A \rightarrow aA$
- $\delta(A, b) = A$
  $A \rightarrow b \quad A$ ⇨ $A \rightarrow bA$
- A is a final state $A \rightarrow \varepsilon$

Scanned with CamScanner

So, the grammar to generate string consisting of any number of a's and b's is given by:



Note: All productions starting from S can be grouped and all productions starting from A can be grouped as shown above. The language can be formally written as:

$$L = \{w : n_a(w) \geq 1, w \in \{a, b\}^*\}$$

Example 9: Obtain grammar to generate string consisting of any number of a's and b's with at least one *b*.

Solution: Same as grammar shown in example 3 but replacing ε by *b*. So, the resultant grammar is:

$$\left.\begin{array}{l} S \to b \\ S \to aS \\ S \to bS \end{array}\right\}$$ **Grammar to generate strings of a's and b's with at least one b**

Example 10: Obtain grammar to generate string consisting of any number of a's and b's with at least one *a* or at least one *b*.

Solution: Same as grammar shown in example 3 but replacing ε by *a* or *b*. So, the resultant grammar is:

$$\left.\begin{array}{l} S \to a \mid b \\ S \to aS \\ S \to bS \end{array}\right\}$$ Note: a | b is read either *a* or *b*.

Grammar to generate strings of a's and b's with at least one b

Example 11: Obtain grammar to accept the following language:

$$L = \{w: |w| \bmod 3 > 0 \text{ where } w \in \{a\}^*\}$$

Solution: The minimum string that can be accepted is either *a* or *aa*. So, the productions to produce *a* or *aa* is given by:

$$S \to a \mid aa$$

Scanned with CamScanner

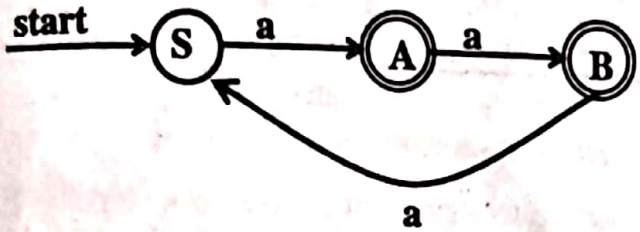
The string aaa should not be accepted. But, aaa followed by *a* or *aa* can be accepted. This is obtained using the production:

$$S \rightarrow aaaS$$

So, the final grammar is given by:

$$S \rightarrow a \mid aa \mid aaaS$$

Note: Alternate method is write a DFA and then obtain the equivalent productions. The DFA is shown below:



$\delta(S, a) = A \Rightarrow S \rightarrow aA$
$\delta(A, a) = B \Rightarrow A \rightarrow aB$
$\delta(B, a) = S \Rightarrow B \rightarrow aS$
A is final state $\Rightarrow A \rightarrow \varepsilon$
B is final state $\Rightarrow B \rightarrow \varepsilon$

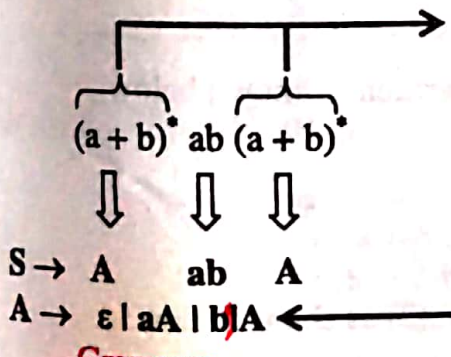
} Grammar

4.4 Grammar from Regular expressions

We have learnt in previous chapter that regular language can also be represented using regular expressions. Now, let us see "How to get the grammar from regular expressions?"

Example 12: Obtain grammar to generate strings of a's and b's having a substring ab

Solution: The regular expression representing strings of a's and b's having a substring is given by:



Represent strings of a's and b's including NULL string. The equivalent productions can be written as:

$A \rightarrow \varepsilon \mid aA \mid bA$ (Example 3, page 4.9)

Scanned with CamScanner

Example 13: Obtain grammar to generate strings of a's and b's ending with string ab

Solution: The regular expression representing strings of a's and b's ending with ab is given by:

$(a + b)^* \, ab$

$(a+b)^*$ → Represent strings of a's and b's including NULL stri
The equivalent productions can be written as:
$A \rightarrow \varepsilon \mid aA \mid bA$ (Example 3, page 4.9)

$S \rightarrow A \; ab$

$A \rightarrow \varepsilon \mid aA \mid bA$

Grammar

Example 14: Obtain grammar to generate strings of a's and b's starting with ab

Solution: The regular expression representing strings of a's and b's having a substring is given by:

$ab \, (a + b)^*$

$(a+b)^*$ → Represent strings of a's and b's including NULL stri
The equivalent productions can be written as:
$A \rightarrow \varepsilon \mid aA \mid bA$ (Example 3, page 4.9)

$S \rightarrow ab \; A$

$A \rightarrow \varepsilon \mid aA \mid bA$

Grammar

Example 15: Obtain grammar to generate the following language:

$$L = \{w : n_a(w) \bmod 2 = 0 \text{ where } w \in \{a, b\}^*\}$$

Solution: String consisting of any number of b's given by the grammar:

$S \rightarrow \varepsilon \mid bS$ ....(1)

(See example 1, page 4.7)

The regular expression for the given language can be written as shown below:

$(b^* \; a \; b^* \; a \; b^*)^*$

$S \rightarrow S \; a \; S \; a \; S$ ....(2)

(from S we can get b*, for details see (1) above)

So, the final grammar is obtained by combining (1) and (2) as shown below:

$S \rightarrow \varepsilon \mid bS \mid SaSaS$

Grammar

4.5 Derivation

Now, let us see "What is derivation?"

Definition: Let $A \rightarrow \alpha B\gamma$ and $B \rightarrow \beta$ are productions in grammar G, where α, β and γ are strings of terminals and/or non-terminals, A and B are non-terminals. The non-terminal A produces the string $\alpha\beta\gamma$ by replacing the non-terminal B in $\alpha B\gamma$ by the string β by applying the production $B \rightarrow \beta$ and can be written as

$$A \Rightarrow \alpha\beta\gamma$$

This process of obtaining string of terminals and/or non-terminals from the start symbol by applying some or all productions is called *derivation*. If a string is obtained by applying only one production, then it is called one-step derivation and is denoted by the symbol '$\Rightarrow$'. If one or more productions are applied to get the string $\alpha\beta\gamma$ from A, then we write

$$A \overset{+}{\Rightarrow} \alpha\beta\gamma$$

If zero or more productions are applied to get the string $\alpha\beta\gamma$ from A, then we write

$$A \overset{*}{\Rightarrow} \alpha\beta\gamma$$

Note: If α is any string of terminals and variables then $\alpha \overset{*}{\Rightarrow} \alpha$ i.e., α derives itself.

Example 16: Consider the grammar shown below from which any arithmetic expression can be obtained.

$E \rightarrow E+E$
$E \rightarrow E-E$
$E \rightarrow E*E$
$E \rightarrow E/E$
$E \rightarrow id$

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.

$E \Rightarrow E+E$
$\Rightarrow id+E$
$\Rightarrow id+E*E$
$\Rightarrow id+id*E$
$\Rightarrow id+id*id$

Scanned with CamScanner

Since the string id + id * id is obtained from the start symbol E by applying more than one production, this can be written as

$$E \overset{+}{\Rightarrow} id + id * id$$

4.5.1 Sentence

Using the grammar, let us see "How to get a sentence?" Before proceeding further let us see "What is a sentence or sentential form?"

Definition: Let G = (V, T, P, S) be a grammar. The string *w* obtained from the grammar G such that $S \overset{+}{\Rightarrow} w$ is called sentence of grammar G. Here, *w* is the string of terminals.

Example 17: In the derivation shown in example 16, id + id * id is the sentence of the grammar. If there is a derivation $S \overset{*}{\Rightarrow} \alpha$, where α contains string of terminals and/or non-terminals, then α is called *sentential form* of G. In the derivation shown in example 16,

E+E, id + E, id + E * E, id + id * E, id + id * id

are all *sentential forms* of the grammar.

4.5.2 Language

A given grammar can generate a set of strings consisting of only of terminals by applying productions in different order. The set of such strings is called the language. Now, let us formally see "What is the language generated by grammar?" The formal definition of the language accepted by a grammar is defined as shown below.

Definition: Let G = (V, T, P, S) be a grammar. The language L(G) generated by the grammar G is

$$L(G) = \{w \mid S \overset{+}{\Rightarrow} w \text{ and } w \in T^*\}$$

i.e., w is a string of terminals (may be ε) obtained from the start symbol S by applying an arbitrary number of productions. The intermediate string of terminals and/or non-terminals obtained during the derivation process is called sentential form of G. The various strings which are the elements of L(G) are called *sentences*.

Example 18: Consider the following grammar

$$S \rightarrow aCa$$
$$C \rightarrow aCa \mid b$$

Scanned with CamScanner

What is the language generated by this grammar?

Solution: Consider the derivation

$S \Rightarrow aCa \Rightarrow aba$ (By applying the 1st and 3rd production)

So, the string aba ∈ L(G)

Consider the derivation

$S \Rightarrow aCa$	By applying $S \rightarrow aCa$
$\Rightarrow aaCaa$	By applying $C \rightarrow aCa$
$\Rightarrow aaaCaaa$	By applying $C \rightarrow aCa$
.	
.	
$\Rightarrow a^nCa^n$	By applying $C \rightarrow aCa$ n-1 times
$\Rightarrow a^nba^n$	By applying $C \rightarrow b$

So, the language L accepted by the grammar G is

$L(G) = \{ a^nba^n \mid n \geq 1 \}$

Scanned with CamScanner

4.6 Grammars for other languages

Example 19: Obtain a grammar to geneare the following language:

$$L = \{a^n b^n : n \geq 0\}$$

Solution: In example 1 (page 4.7) we have the following grammar:

$$S \rightarrow \varepsilon \mid aS$$

that generates string consisting of any number of a's. Now, for every *a* one *b* has to be generated. This is obtained by suffixing aS with a *b*. So, the grammar to generate the given language is:

$$S \rightarrow \varepsilon \mid aSb$$

Example 20: Obtain a grammar to generate the following language:

$$L = \{a^n b^n : n \geq 1\}$$

Solution: It is similar to the previous example. But, instead of generating at least ε, we should generate at least *ab*. This is achieved by replacing ε by *ab*. So, the final grammar is given by:

$$S \rightarrow ab \mid aSb$$

Scanned with CamScanner

Example 21: Obtain a grammar to geneare the following language:

$$L = \{a^{n+1}b^n : n \geq 0\}$$

Solution: It is similar to the example 19. But, one extra *a* should be generated. This is achived by replacing ε with *a*. So, the final grammar to generate the given language is:

$$S \rightarrow a \mid aSb$$

Example 22: Obtain a grammar to geneare the following language:

$$L = \{a^n b^{n+1} : n \geq 0\}$$

Solution: It is similar to the example 19. But, one extra *b* should be generated. So, the final grammar to generate the given language is:

$$S \rightarrow b \mid aSb$$

Example 23: Obtain a grammar to geneare the following language:

$$L = \{a^n b^{n+2} : n \geq 0\}$$

Solution: It is similar to the example 19. But, two extra *b's* should be generated. So, the final grammar to generate the given language is:

$$S \rightarrow bb \mid aSb$$

Example 24: Obtain a grammar to geneare the following language:

$$L = \{a^n b^{2n} : n \geq 0\}$$

Solution: Consider the grammar shown in example 19:

$$S \rightarrow \varepsilon \mid aSb$$

The above grammar generates the language $a^n b^n$. But, we want $a^n b^{2n}$ i.e., for every *a* we want two generate *b*'s. This is achived by suffix one more *b* to the above grammar. So, the final grammar is:

$$S \rightarrow \varepsilon \mid aSbb$$

Example 25: Let $\Sigma = \{a, b\}$. Obtain a grammar G generating set of all palindromes over Σ.

Solution: The recursive definition of a palindrome along with corresponding productions is shown below:

1. ε is a palindrome. The equivalent productions is $S \rightarrow \varepsilon$
2. *a* and *b* are palindromes. The equivalent productions are $S \rightarrow a \mid b$

Scanned with CamScanner

3. If *w* is a palindrome then the string *awa* and the string *bwb* are palindromes. The equivalent productions are

$$S \rightarrow aSa \mid bSb$$

So, the grammar to generate strings of palindromes over Σ is given by G = (V, T, P, S) where

V = {S}
T = { a, b}
P = {

S	→	ε	[Definition 1]
S	→	a | b	[Definition 2]
S	→	aSa | bSb	[Definition 3]

}
S is the start symbol

Example 26: Obtain a grammar to geneare the following language:

$$L = \{ww^R \text{ where } w \in \{a, b\}^*\}$$

Solution: The language can be written as:

$$L = \{aa, bb, abba, baab, aaaa, bbbb, \ldots.\}$$

Observe that the given string is a palindrome of even length. This is achived by deleting the productions S → a | b. So, the final grammar is given by:

$$S \rightarrow \varepsilon$$
$$S \rightarrow aSa \mid bSb$$

Note: In the above grammar if the production

$$S \rightarrow \varepsilon$$

is replaced by

$$S \rightarrow c$$

the resulting grammar will generate the language m $L = \{wcw^R \mid w \in \{a,b\}^*\}$

Example 27: Obtain a grammar to generate a language consisting of all non-palindromes over {a, b}

Solution: When we scan from left to right and right to left simultaneously, we may find same symbols for a while, but at some point while scanning we may find

Scanned with CamScanner

a symbol on the left which is different from the symbol on the right. Then the given string is not a palindrome. The corresponding S-productions for this can be

$$S \rightarrow aSa \mid bSb \mid A$$

where A generates string of only non palindromes. To get a string of non-palindrome, the string derivable from A should have different symbols in the beginning and in the end, the length of which should be greater than or equal to 2 and so the A-productions can take the following form

$$A \rightarrow aBb \mid bBa$$

From production B, if we can generate any string of a's and b's including ε i.e., $(a+b)^*$, the string derivable from A is still a non-palindrome. So, the B-productions can be written as

$$B \rightarrow aB \mid bB \mid \varepsilon$$

So, the complete grammar to generate strings of non-palindromes is given by G = (V, T, P, S) where

V = { S, A, B }
T = { a, b }
P = {

S	→	aSa | bSb	[Generates palindromes both on left side and right side]
S	→	A	[Generates a non-palindrome]
A	→	aBb | bBa	[Generates a non palindrome with B generating any number of a's and b's]
B	→	aB | bB | ε	[Generates any combination of a's and b's]

}
S is the start symbol

The derivation for the string ababba which is not a palindrome is shown below.

$S \Rightarrow aSa$
$\Rightarrow abSba$
$\Rightarrow abAba$
$\Rightarrow abaBba$
$\Rightarrow ababba$

Example 27: Obtain the grammar to generate the language

$$L = \{ 0^m 1^m 2^n \mid m \geq 1 \text{ and } n \geq 0 \}$$

Scanned with CamScanner

Solution: Given the language the productions can be generated as shown below:

$$L = \{ 0^m 1^m 2^n \mid m \geq 1 \text{ and } n \geq 0 \}$$

$$S \rightarrow A \quad B \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(1)$$

where

- The variable A should produce *m* number of 0's followed by *m* number of 1's with a minimum string 01 (Since $m \geq 1$). This is achieved using the following production:

  $A \rightarrow 01 \mid 0A1$ [Similar to example 20, page 4.17]

- B should produce any number of 2's. Any number of 2's can be generated using the production:

  $B \rightarrow \varepsilon \mid 2B$ [Similar to example 1, page 4.7]

So, final grammar to accept the given language is:

$$\left.\begin{array}{l} S \rightarrow A \quad B \\ A \rightarrow 01 \mid 0A1 \\ B \rightarrow \varepsilon \mid 2B \end{array}\right\} \text{Grammar to accept } L = \{0^m 1^m 2^n \mid m \geq 1 \text{ and } n \geq 0 \}$$

The following grammar also generates the same language. The reader is required to verify the answer.

$$S \rightarrow A \mid S2$$
$$A \rightarrow 01 \mid 0A1$$

Example 28: Obtain the grammar to generate the language

$$L = \{w \mid n_a(w) = n_b(w)\}$$

Note: $n_a(w) = n_b(w)$ means, number of a's in the string *w* should be equal to number of b's in the string *w*. To get equal number of a's and b's, we know that there are three cases:

1. An empty string denoted by ε has equal number of a's and b's (i.e., zero a's and zero b's).
2. The symbol 'a' can be followed by the symbol 'b'
3. The symbol 'b' can be followed by the symbol 'a'

The corresponding productions for these three cases can be written as

Scanned with CamScanner

$S \rightarrow \varepsilon$
$S \rightarrow aSb$
$S \rightarrow bSa$

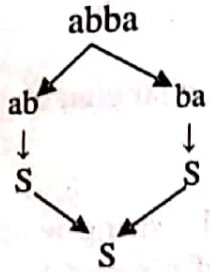
Using these productions the strings of the form ε, ab, ba, abab, baba etc., can be generated. But, the strings such as abba, baab, etc., where the string starts and ends with the same symbol, can not be generated from these productions (even though they are valid strings). So, to obtain the productions to generate such strings, let us divide the string into two substrings. For example, let us take the string 'abba'. This string can be split into two substrings 'ab' and 'ba'. The substring 'ab' can be generated from S and the derivation is shown below:

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

Similarly, the substring 'ba' can be generated from S and the derivation is shown below:

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

i.e., the first sub string 'ab' can be generated from S as shown in the first derivation and the second sub string 'ba' can also be generated from S as shown in second derivation. So, To get the string 'abba' from S, perform the derivation in reverse order as shown below:



So, to get a string such that it starts and ends with the same symbol, the production to be used is

$S \rightarrow SS$

The final grammar to generate the language $L = \{w \mid n_a(w) = n_b(w)\}$ is $G = (V, T, P, S)$ where

Scanned with CamScanner

V = {S}
T = {a, b}
P = {

$S \rightarrow \epsilon \mid aSb \mid bSa$.

$S \rightarrow SS$.

}

S is the start symbol

Example 29: What is the language generated by the grammar

$S \rightarrow 0A \mid \varepsilon$
$A \rightarrow 1S$

Solution: 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

$S \Rightarrow 0A$ (By applying $S \rightarrow 0A$)
$\Rightarrow 01S$ (By applying $A \rightarrow 1S$)
$\Rightarrow 010A$ (By applying $S \rightarrow 0A$)
$\Rightarrow 0101S$ (By applying $A \rightarrow 1S$)
$\Rightarrow 0101$ (By applying $S \rightarrow \varepsilon$)

So, alternatively applying the productions $S \rightarrow 0A$ and $A \rightarrow 1S$ and finally applying the production $S \rightarrow \varepsilon$, we get string consisting of only of 01's. So, both null string i.e., ε and string consisting of 01's can be generated from this grammar. So, the language generated by this grammar is

$L = \{w \mid w \in \{01\}^*\}$ or $L = \{(01)^n \mid n \geq 0\}$

Note: The above language can also be generated from the following two grammars:

$S \rightarrow 01S \mid \varepsilon$

or

$S \rightarrow S01 \mid \varepsilon$

Scanned with CamScanner

Example 30: Obtain a CFG to generate a string of balanced parentheses.

Solution: The grammar G to generate a string of balanced parentheses is given by G = (V, T, P, S) where

$V = \{S\}$
$T = \{ (,), [,], \{, \} \}$
$P = \{$
$S \rightarrow (S)$
$S \rightarrow [S]$
$S \rightarrow \{S\}$
$S \rightarrow SS$
$S \rightarrow \varepsilon$
$\}$
S is the start symbol

Example 31: Obtain a grammar to generate the language
$L = \{0^i1^j \mid i \neq j, i \geq 0 \text{ and } j \geq 0\}$

Note: It is clear from the statement that if a string has n number of 0's as the prefix, this prefixed string should not be followed by n number of 1's i.e., we should not have equal number of 0's and 1's. At the same time 0's should precede 1's. The grammar for this can be written as

G = (V, T, P, S) where

$V = \{S, A, B, C\}$
$T = \{0, 1\}$
$P = \{$

$S \rightarrow BA \mid$		[At least one 0 is preceded by 0^n1^n]
$S \rightarrow AC$		[At least one 1 is followed by 0^n1^n]
$A \rightarrow 0A1 \mid \varepsilon$		[Generates $0^n1^n \mid n \geq 0$]
$B \rightarrow 0B \mid 0$		[At least one 0 is generated]
$C \rightarrow 1C \mid 1$		[At least one 1 is generated]

$\}$
S is the start symbol

Note: The following grammar also generates the language $L = \{0^i1^j \mid i \neq j, i \geq 0 \text{ and } j \geq 0\}$

Scanned with CamScanner

V = {S, A, B, C}
T = {0, 1}
P = {

S	→	0S1	[Generates 0^n1^n recursively]
S	→	A	[To generate more 0's than 1's]
S	→	B	[To generate more 1's than 0's]
A	→	0A | 0	[At least one 0 is generated]
B	→	1B | 1	[At least one 1 is generated]

}
S is the start symbol

Example 32: Obtain a grammar to generate the language
$L = \{a^{n+2}b^m \mid n \geq 0 \text{ and } m > n\}$.

Solution: It is clear from the above statement that the set of strings that can be generated by this language can be represented as

$L = \{aabb^*, aaabbb^*, aaaabbbb^*, \ldots\ldots\ldots\}$

Observe that the language consists of a string $a^nb^n \mid n \geq 1$ preceded by one *a* and followed by zero or more *b*'s. This prompts us to have a production of the form

$S \rightarrow aAB$

where the language $a^nb^n \mid n \geq 1$ is generated from the variable A and zero or more b's are generated from the variable B. The language $a^nb^n \mid n \geq 1$ is generated from the variable A using the productions shown below

$A \rightarrow aAb \mid ab$

and zero or more *b*'s are generated from the production

$B \rightarrow bB \mid \varepsilon$

So, the final grammar G = (V, T, P, S) which can generate the given language is

S	→	aAB	['a' followed by a string having n number of a's followed by n number of b's followed by zero or more b's]
A	→	aAb | ab	[generates a^nb^n]
B	→	bB | ε	[Generates zero or more b's]

Scanned with CamScanner

Example 33: Obtain a grammar to generate the language

$$L = \{a^n b^m \mid n \geq 0, m > n\}$$

Solution: It is clear from the above statement that the set of strings that can be generated by this language can be represented as

$$L = \{\underset{m \geq 1}{\varepsilon bb^*}, \underset{m \geq 2}{abbb^*}, \underset{m \geq 3}{aabbbb^*}, \ldots\ldots\ldots\}$$

where b^* represents zero or more b's. Observe that the language consists of a string $a^n b^n \mid n \geq 0$ followed by one or more b's. This prompts us to have a production of the form

$$S \rightarrow AB$$

where the language $a^n b^n \mid n \geq 0$ is generated from the variable A and one or more b's are generated from the variable B. The language $a^n b^n \mid n \geq 0$ is generated from the variable A using the productions shown below

$$A \rightarrow aAb \mid \varepsilon$$

and one or more b's are generated from the production

$$B \rightarrow bB \mid b$$

So, the final grammar $G = (V, T, P, S)$ which can generate the given language is

$V = \{S, A, B\}$

$T = \{a, b\}$

$P = \{$

$S \rightarrow AB$ [Generates $a^n b^n \mid n \geq 0$]

$A \rightarrow aAb \mid \varepsilon$ [Generates one or more b's]

$B \rightarrow bB \mid b$

$\}$

S - is the start symbol

Note: The same language can also be generated from the following grammar:

$S \rightarrow aSb \mid B$ [Generates $a^n b^n \mid n \geq 0$ followed by at least one b]

$B \rightarrow bB \mid b$ [Generates one or more b's]

Scanned with CamScanner

Example 34: Obtain a grammar to generate the language

$L = \{a^n b^{n-3} \mid n \geq 3\}$

Note: It is clear from the above statement that the set of strings that can be generated by this language can be represented as

$L = \{aaa, aaaab, aaaaabb, aaaaaabbb, \ldots\ldots\}$

It is observed from the above set that the string 'aaa' is followed by the string $a^n b^n \mid n \geq 0$. So, the first production that we can think of is

$S \rightarrow aaaA$

where the string $a^n b^n \mid n \geq 0$ can be obtained using A as shown below:

$A \rightarrow aAb \mid \varepsilon$

So, the final grammar that can generate the given language is

$V = \{S, A\}$
$T = \{a,b\}$
$P = \{$
$S \rightarrow aaaA$
$A \rightarrow aAb \mid \varepsilon$
$\}$
S - is the start symbol

Note: The same language can also be generated from the grammar with the following productions:

$S \rightarrow aSb \mid aaa$

Example 35: Obtain a grammar to generate the language

$L = L_1 L_2$

where

$L_1 = \{a^n b^m \mid n \geq 0, m > n\}$
$L_2 = \{0^n 1^{2n} \mid n \geq 0\}$

Solution: The grammar corresponding to the language L_1 is already obtained in example 33. For convenience it is provided again

$S_1 \rightarrow aS_1b \mid bB$
$B \rightarrow bB \mid b$

Scanned with CamScanner

The grammar corresponding to the language L_2 is already obtained in example 24. For convenience it is provided again

$V = \{S_2\}$
$T = \{0, 1\}$
$P = \{$
$S_2 \rightarrow 0S_211 \mid \varepsilon$
$\}$
S_2 is the start symbol

The resulting language

$$L = L_1L_2$$

can be generated from the grammar obtained by concatenating the start symbol S_1 of first grammar with the start symbol S_2 of the second grammar as

$$S \rightarrow S_1S_2$$

where S is the start symbol of the resultant grammar. So, the final grammar which accepts

$$L = L_1L_2$$

is shown below:

$V = \{S, S_1, S_2, B\}$
$T = \{a,b,0,1\}$
$P = \{$
$S \rightarrow S_1S_2$
$S_1 \rightarrow aS_1b \mid bB$
$S_2 \rightarrow 0S_211 \mid \varepsilon$
$B \rightarrow bB \mid b$
$\}$
S_1 - is the start symbol

Example 36: Obtain a grammar to generate the language

$$L = L_1 \cup L_2$$

where

$L_1 = \{a^nb^m \mid n \geq 0, m > n\}$
$L_2 = \{0^n1^{2n} \mid n \geq 0\}$

Note: This problem is similar to the previous problem, except that from the start symbol S either S_1 is derived or S_2 is derived as shown below.

$$S \rightarrow S_1 \mid S_2$$

Scanned with CamScanner

The rest of the productions remain same. The final grammar is shown below (Both the grammars below generate the same language)

$V = \{ S, S_1, S_2, A, B\}$
$T = \{a,b,0,1\}$
$P = \{$
$S \rightarrow S_1 \mid S_2$
$S_1 \rightarrow AB$
$S_2 \rightarrow 0S_211 \mid \varepsilon$
$A \rightarrow aAb \mid \varepsilon$
$B \rightarrow bB \mid b$
$\}$
S_1 - is the start symbol

$V = \{ S, S_1, S_2, B\}$
$T = \{a,b,0,1\}$
$P = \{$
$S \rightarrow S_1 \mid S_2$
$S_1 \rightarrow aS_1b \mid bB$
$S_2 \rightarrow 0S_211 \mid \varepsilon$
$B \rightarrow bB \mid \varepsilon$
$\}$
S_1 - is the start symbol

Example 37: Obtain a grammar to generate the language
$L = \{w : |w| \bmod 3 \neq |w| \bmod 2\}$ on $\Sigma = \{a\}$

Solution: The following grammar also accepts the same language

$V = \{S\}$
$T = \{a\}$
$P = \{$
$S \rightarrow aa \mid aaa \mid aaaa \mid aaaaa \mid aaaaaS$
$\}$
S - is the start symbol

Note: Another easy method and totally different grammar can be obtained by drawing the DFA and then converting it into grammar. (See the DFA shown in chapter 1)

Example 38: Obtain a grammar to generate the language
$L = \{w : |w| \bmod 3 \geq |w| \bmod 2\}$ on $\Sigma = \{a\}$

Solution: The grammar to accept the above language is shown below:

$V = \{S\}$
$T = \{a\}$
$P = \{$
$S \rightarrow \varepsilon \mid a \mid aa \mid aaaa \mid aaaaa \mid aaaaaS$
$\}$
S - is the start symbol

Scanned with CamScanner

Note: Another easy method and totally different grammar can be obtained by drawing the DFA and then converting it into grammar. (See the DFA shown in chapter 1)

Example 39: Obtain a grammar to generate set of all strings with exactly one *a* when $\Sigma = \{a, b\}$

Since w ∈ L should have exactly one a, this single *a* can be preceded by any number of *b*'s which can be achieved using the production

$$S \rightarrow bS \mid aB$$

Note that each time bS is substituted in place of S, one extra *b* is generated. Finally, if S is replaced by aB, we will have exactly one *a* followed by a non-terminal B. From this B, we should generate any number of *b*'s and can be achieved using the production

$$B \rightarrow bB \mid \varepsilon$$

So, the final grammar to generate at least one *a* is

$V = \{S, B\}$
$T = \{a, b\}$
$P = \{$
$S \rightarrow bS \mid aB$
$B \rightarrow bB \mid \varepsilon$
$\}$
S - is the start symbol

Example 40: Obtain a grammar to generate set of all strings with at least one *a* when $\Sigma = \{a, b\}$

Solution: String consisting of at least one *a* implies one or more *a*'s. These one or more *a*'s can be preceded by any number of *b*'s which can be achieved using the production

$$S \rightarrow bS$$

Scanned with CamScanner

Note that each time bS is substituted in place of S, one extra *b* is generated. Once there are no *b*'s, we should be in a position to generate at least one *a* from S and can be generated by the production

$$S \rightarrow aA$$

Once one *a* is generated, this *a* can be followed by any number of *a*'s and/or *b*'s. The productions to generate such *a*'s and *b*'s are

$$A \rightarrow aA \mid bA \mid \varepsilon$$

So, the final grammar to generate at least one *a* is

$V = \{S, A\}$
$T = \{a, b\}$
$P = \{$
$S \rightarrow bS \mid aA$
$A \rightarrow aA \mid bA \mid \varepsilon$
$\}$
S - is the start symbol

Example 41: Obtain a grammar to generate the set of all strings with no more than three *a*'s when $\Sigma = \{a, b\}$

Solution: String containing note more than three *a*'s implies that

1. there can be no *a*'s at all
2. there can be only one *a*
3. there can be only two *a*'s
4. there can be only three *a*'s

But, at any point of time number of *a*'s should not exceed 3 and the string can have any number of *b*'s. Let us take all these four cases one by one.

Case 1: there can be no *a*'s at all, but there is no restriction on the number of *b*'s. The production corresponding to this is

$$S \rightarrow bS \mid \varepsilon$$

Scanned with CamScanner

Case 2: There can be only one *a*. The single *a* can be obtained by the start symbol S using the production

$$S \rightarrow aA$$

After applying this production, we get one *a* followed by a non-terminal A. From this we can generate any number of *b*'s using the production

$$A \rightarrow bA \mid \varepsilon$$

and the string can be accepted. After the input of some *b*'s, we can input one more *a* which is the next case.

Case 3: There can be two *a*'s. When we get the non-terminal A, we would have got one *a*. To get one more *a*, we can have the production

$$A \rightarrow aB$$

After applying this production, we would have got some number of *b*'s and exactly two *a*'s where the second *a* is followed by a non-terminal B. Any number of *b*'s can be generated from B and the corresponding productions are:

$$B \rightarrow bB \mid \varepsilon$$

After applying this production some number of times, we have two *a*'s followed by zero or more *b*'s. After some *b*'s, we can input the third *a* which also should be accepted and the corresponding production is

$$B \rightarrow aC$$

After applying this production, we have accepted three *a*'s which can be followed by any number of *b*'s. The corresponding productions are

$$C \rightarrow bC \mid \varepsilon$$

Now we have accepted maximum of three *a*'s and no more *a*'s can be accepted thus producing not more than three *a*'s. So, the final grammar is

$$\begin{aligned} S &\rightarrow bS \mid aA \mid \varepsilon \\ A &\rightarrow bA \mid aB \mid \varepsilon \\ B &\rightarrow bB \mid aC \mid \varepsilon \\ C &\rightarrow bC \mid \varepsilon \end{aligned}$$

Scanned with CamScanner

Note: For the language another grammar can be generated as shown below: String containing note more than three *a*'s implies that

1. there can be no *a*'s at all. The corresponding production can be S → B
2. there can be only one *a*. The corresponding production is S → BaB
3. there can be only two *a*'s. The corresponding production is S → BaBaB
4. there can be only three *a*'s. The corresponding production is S → BaBaBaB

Now, the string containing any number of b's can be generated from the production B → bB | ε. So, the final grammar to generate the given language is given by G = (V, T, P, S) where

V = {S, B}
T = {a, b}
P = {
S → B | BaB | BaBaB | BaBaBaB
B → bB | ε
}
S - is the start symbol

Example 42: Obtain a grammar to generate the language $L = \{w \mid n_a(w) = n_b(w) + 1\}$

Solution: The problem is similar to example 1.9, except that w ∈ L should have one more *a* either in the beginning or at the end or at the middle. This can be achieved using the production

S → AaA

where A generates equal number of *a*'s and *b*'s using the productions

A → aAb
A → bAa
A → AA
A → ε

So, the final grammar to generate the language

$$L = \{w \mid n_a(w) = n_b(w) + 1\}$$

is shown below:

S → AaA
A → aAb | bAa | AA | ε

Scanned with CamScanner

Example 43: Obtain the grammar to generate the language

$$L = \{w \mid n_a(w) > n_b(w)\}$$

Solution: We have already seen that the following grammar produces equal number of *a*'s and *b*'s:

$$A \rightarrow aAb$$
$$A \rightarrow bAa$$
$$A \rightarrow AA$$
$$A \rightarrow \varepsilon$$

Since number of a's should be greater than number of b's, w should be in a position to generate 1 or more a's. One or more a's can be generated using the production:

$$B \rightarrow aB \mid a$$

Since more number of *a*'s can occur in the beginning, at the end or at the middle, we should have a production to generate as many *a*'s as possible in the respective places. Equal number of a's and b's can be followed by one or more a's which can be achieved by introducing the production

$$S \rightarrow AB$$

Equal number of a's and b's can be preceded by one or more a's which can be achieved by introducing the production

$$S \rightarrow BA$$

Equal number of a's and b's can have one or more a's in the middle which can be achieved by introducing the production

$$S \rightarrow ABA$$

So, the final grammar to generate the language

$$L = \{w \mid n_a(w) > n_b(w)\}$$

is G = (V, T, P, S) where

Scanned with CamScanner

$V = \{S, A, B\}$
$T = \{a, b\}$
$P = \{$
$S \rightarrow AB|BA|ABA$
$A \rightarrow aAb$
$A \rightarrow bAa$
$A \rightarrow AA$
$A \rightarrow \varepsilon$
$B \rightarrow aB \mid a$
$\}$
S is the start symbol

Example 44: Obtain a grammar to generate a language of strings of 0's and 1's having a substring 000

i.e., $L = \{w \mid w \in \{0,1\}^*$ with at least one occurrence of '000'$\}$

Solution: It is clear from this definition that the substring 000 can be preceded and followed by strings of 0's and 1's of any length which can be generated using the production of the form

$$S \rightarrow A000A$$

where any strings of 0's and 1's are generated from the non-terminal A using the productions

$$A \rightarrow 0A \mid 1A \mid \varepsilon$$

So, the final grammar to generate the given language is $G = (V, T, P, S)$ where

$V = \{S\}$
$T = \{0,1\}$
$P = \{$
$S \rightarrow A000A$
$A \rightarrow 0A \mid 1A \mid \varepsilon$
$\}$
S is the start symbol

Example 45: Obtain a grammar to generate the following language
i.e., $L = \{a^n b^m c^k \mid n + 2m = k$ for $n \geq 0, m \geq 0\}$

Scanned with CamScanner

Solution: It is required to generate the grammar for the language

$$L = \{a^n b^m c^k \mid n + 2m = k \text{ for } n \geq 0, m \geq 0\}$$

Let us express k in terms of n and m in $a^n b^m c^k$. So, after substituting for k using $k = n + 2m$ we have:

$$L = \{a^n b^m c^{n+2m} \mid n \geq 0, m \geq 0\}$$

which is equivalent to

$$L = \{a^n b^m c^{2m} c^n \mid n \geq 0, m \geq 0\}$$

So, it is clear from these examples that the given language can be expressed as:

$$L = \{w \mid a^n b^m c^{2m} c^n \text{ where } m \geq 0, n \geq 0\}$$

The sub string $b^m c^{2m}$ can be generated from the following productions:

$$A \rightarrow bAcc \mid \varepsilon$$

The substring thus generated from A-production is enclosed between a^n and c^n and the corresponding productions can be written as

$$S \rightarrow aSc \mid A$$

So, the final grammar to generate the given language is G = (V, T, P, S) where

V = {S}
T = {a,b,c}
P = {
S → aSc | A
A → bAcc | ε
}
S is the start symbol

Example 46: Obtain a grammar to generate integers

Solution: The sign of a number can be '+' or '–' or ε. The production for this can be written as

$$S \rightarrow + \mid - \mid \varepsilon$$

In the above production, if ε is derived from S, the sign for a number will not be generated. A number can be formed from any of the digits 0,1,2,....9. The production to obtain these digits can be written as

$$D \rightarrow 0 \mid 1 \mid 2 \mid \ldots\ldots \mid 9$$

Scanned with CamScanner

A number N can be recursively defined as follows.

1. A digit is a number (i.e., N → D)
2. The number followed by a digit is a number (i.e., N → ND)
   or
   a digit followed by number is also a number (i.e., N → DN)

The productions for this recursive definition can be written as

$$N \rightarrow D$$
$$N \rightarrow ND \mid DN$$

An integer number I can be a number N or a sign (an optional plus and a minus) followed by number N. The production for this can be written as

$$I \rightarrow N \mid SN$$

So, the grammar G to obtain integer numbers can be written as

G = (V,T,P,S) where

V = { D, S, N, I }
T = { +, -, 0, 1, 2, ...9 }
P = {

I	→	N | SN	[Generate signed or unsigned number]
N	→	D | ND | DN	[Generates one or more digits]
S	→	+ | - | ε	[Generate the sign]
D	→	0 | 1 | 2 |.....|9	[Generate the digits]

}
S = I which is the start symbol

The unsigned number 1965 and signed number +1965 can be derived as shown below:

$$\begin{aligned} I &\Rightarrow N \\ &\Rightarrow ND \\ &\Rightarrow N5 \\ &\Rightarrow ND5 \\ &\Rightarrow N65 \\ &\Rightarrow ND65 \\ &\Rightarrow N965 \\ &\Rightarrow D965 \\ &\Rightarrow 1965 \end{aligned}$$

$$\begin{aligned} I &\Rightarrow SN \\ &\Rightarrow +N \\ &\Rightarrow +ND \\ &\Rightarrow +N5 \\ &\Rightarrow +ND5 \\ &\Rightarrow +N65 \\ &\Rightarrow +ND65 \\ &\Rightarrow +N965 \\ &\Rightarrow +D965 \\ &\Rightarrow +1965 \end{aligned}$$

Scanned with CamScanner

Example 47: Let G = (V,T,P,S) be a CFG where

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

What is the language generated by this grammar?

Solution: The null string ε can be obtained by applying the production $S \rightarrow \varepsilon$ 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.

Scanned with CamScanner

Example 48: 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 shown below:

Scanned with CamScanner

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 = E 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

4.7 Leftmost derivation

Example 49: 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.

Scanned with CamScanner

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

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

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

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. Now, let us see "What is left most derivation?" 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 49. It is clear from leftmost derivation that each of the following:

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

can be obtained from the start symbol. Each string in the set is called the sentential form.

Now, let us see "What is 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

Scanned with CamScanner

the derivation process only a right most variable is replaced, then α is called *right-sentential* form.

Example 50: Obtain the leftmost derivation for the string aaabbabbba using the following grammar.

$$S \rightarrow aB \mid bA$$
$$A \rightarrow aS \mid bAA \mid a$$
$$B \rightarrow bS \mid aBB \mid b$$

The leftmost derivation for the string aaabbabbba is shown below:

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

4.8 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 49 is shown below.

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

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

Scanned with CamScanner

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

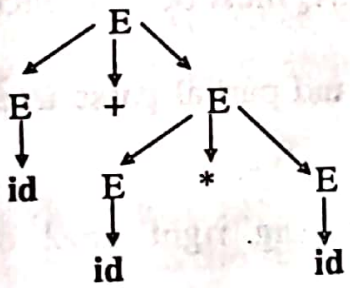
4.9 Derivation Tree (Parse tree)

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. Now, let us see "What is derivation tree or parse tree?" 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 $X_1, X_2, X_3, \ldots X_n$ are all children of A from left, then $A \rightarrow X_1X_2X_3\ldots X_n$ must be a production in P.

The parse tree for the right most derivation shown in section 4.8 is shown below:



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

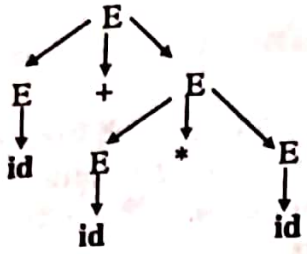
id +id*id

is called the *yield* of the tree. Now, let us see "What is the yield of the tree?" The, 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.

Scanned with CamScanner

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. Now, let us see "What is partial derivation 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

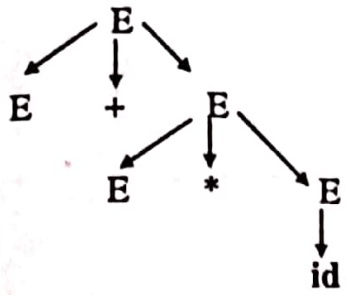
$$E \rightarrow E + E \mid E * E \mid id$$

is shown below

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

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

Scanned with CamScanner



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

4.10 Ambiguous grammar

In this section, let us see "What is 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 51: Consider the grammar shown below from which any arithmetic expression can be obtained.

Scanned with CamScanner

$$E \rightarrow E+E$$
$$E \rightarrow E-E$$
$$E \rightarrow E*E$$
$$E \rightarrow E/E$$
$$E \rightarrow (E) \mid I$$
$$I \rightarrow id$$

Show that the grammar is ambiguous.

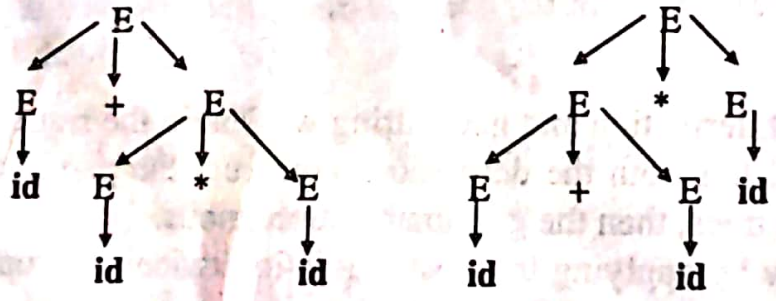
Note: Leftmost derivation, rightmost derivation and parse trees are equivalent.

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

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

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

The corresponding derivation trees for the two leftmost derivations are shown below:



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

Example 52: Is the following grammar ambiguous?

$$S \rightarrow aS \mid X$$
$$X \rightarrow aX \mid a$$

Consider the two leftmost derivations for the string aaaa.

Scanned with CamScanner

$S \Rightarrow aS$
$\Rightarrow aaS$
$\Rightarrow aaaS$
$\Rightarrow aaaX$
$\Rightarrow aaaa$

$S \Rightarrow X$
$\Rightarrow aX$
$\Rightarrow aaX$
$\Rightarrow aaaX$
$\Rightarrow aaaa$

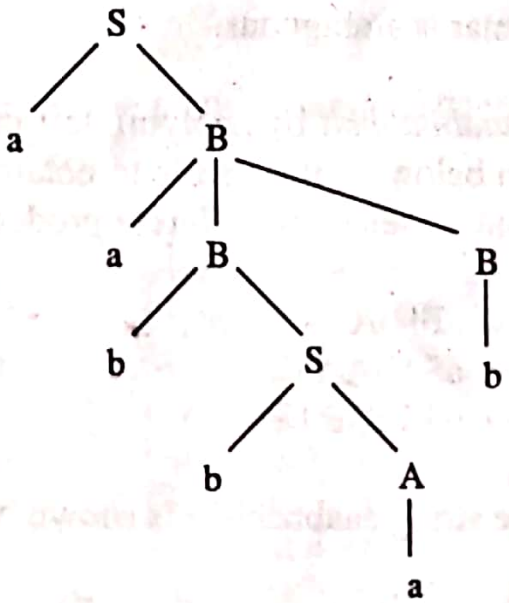
Since there are two leftmost derivations for the same sentence aaaa, the given grammar is ambiguous.

Example 53: Is the following grammar ambiguous?

$S \rightarrow aB \mid bA$
$A \rightarrow aS \mid bAA \mid a$
$B \rightarrow bS \mid aBB \mid b$

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

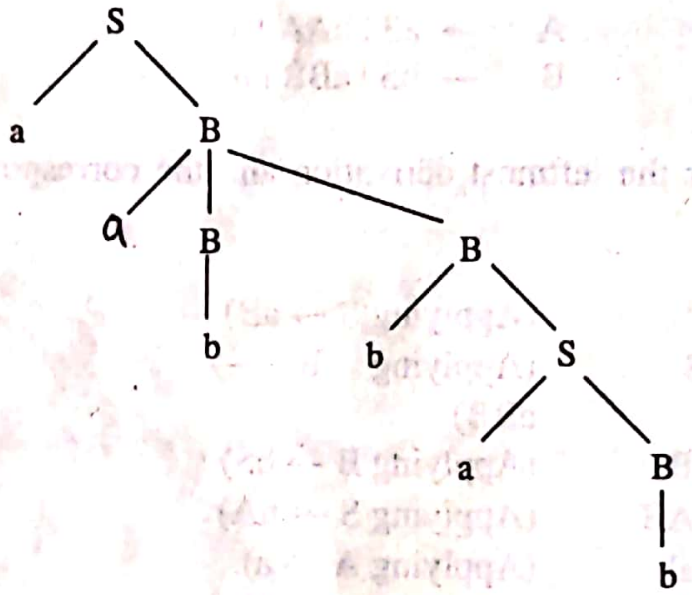
$S \Rightarrow aB$ (Applying $S \rightarrow aB$)
$\Rightarrow aaBB$ (Applying $B \rightarrow aBB$)
$\Rightarrow aabSB$ (Applying $B \rightarrow bS$)
$\Rightarrow aabbAB$ (Applying $S \rightarrow bA$)
$\Rightarrow aabbaB$ (Applying $A \rightarrow a$)
$\Rightarrow aabbab$ (Applying $B \rightarrow b$)



Scanned with CamScanner

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 54: 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?

$S \rightarrow aB \mid bA$
$A \rightarrow aS \mid bAA \mid a$
$B \rightarrow bS \mid aBB \mid b$

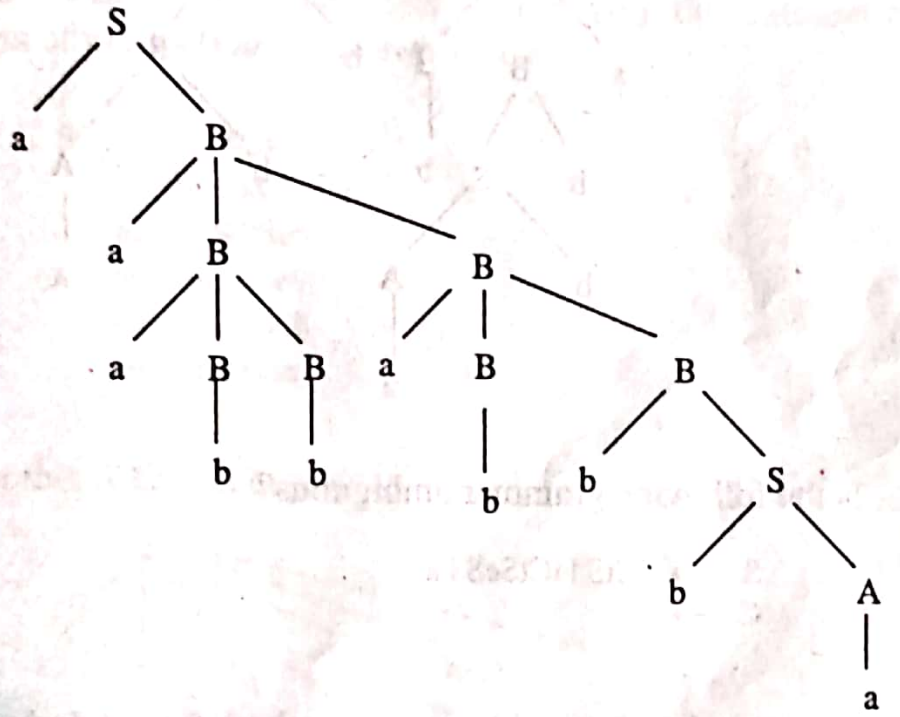
The leftmost derivation for the string aaabbabbba is shown below:

$S \Rightarrow aB$ (Applying $S \rightarrow aB$)
$\Rightarrow aaBB$ (Applying $B \rightarrow aBB$)

Scanned with CamScanner

$\Rightarrow$ aaaBBB (Applying B $\rightarrow$ aBB)
$\Rightarrow$ aaabBB (Applying B $\rightarrow$ b)
$\Rightarrow$ aaabbB (Applying B $\rightarrow$ b)
$\Rightarrow$ aaabbaBB (Applying B $\rightarrow$ aBB)
$\Rightarrow$ 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 the above derivation is shown below:

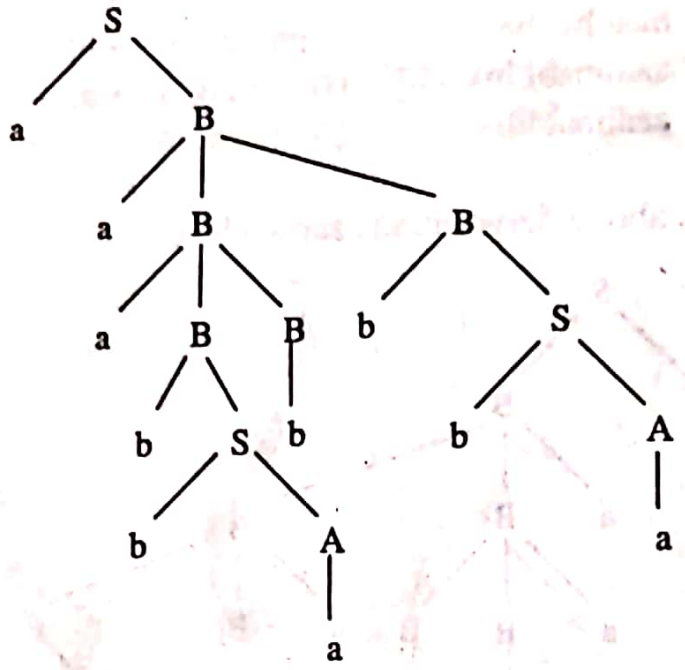


The leftmost derivation for the same string aaabbabbba but by applying different set of productions is shown below:

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)

Scanned with CamScanner

The parse tree for this derivation is shown in figure below:



Example 55: Is the following grammar ambiguous?

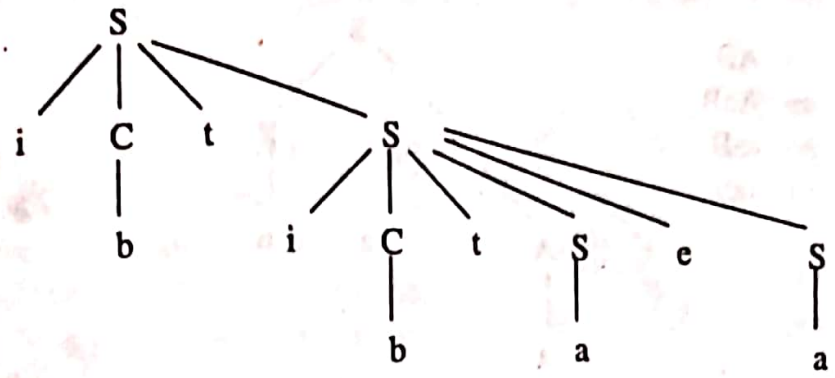
$$S \rightarrow iCtS \mid iCtSeS \mid a$$
$$C \rightarrow b$$

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:

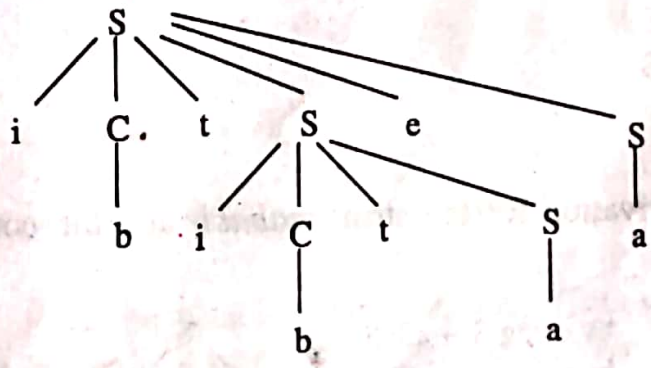
Scanned with CamScanner



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.

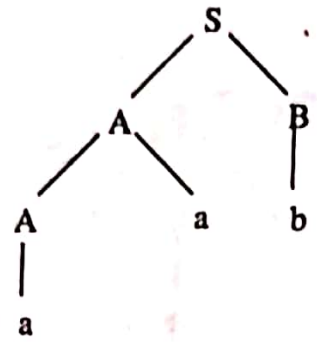
Example 56: Is the grammar ambiguous?

$$
\begin{aligned}
S &\rightarrow AB \mid aaB \\
A &\rightarrow a \mid Aa \\
B &\rightarrow b
\end{aligned}
$$

Scanned with CamScanner

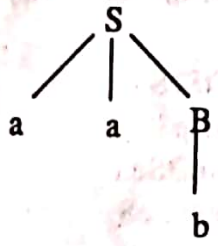
Consider the left most derivation for the string *aab* and the corresponding pares tree

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



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

$S \Rightarrow aaB$
$\Rightarrow aab$



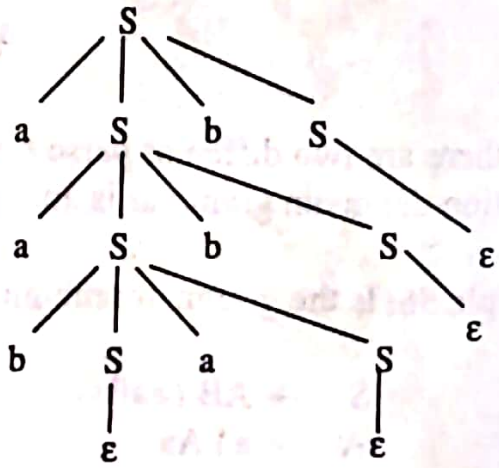
Since there are two parse trees for the string *aab*, the given grammar is ambiguous.

Example 57: 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

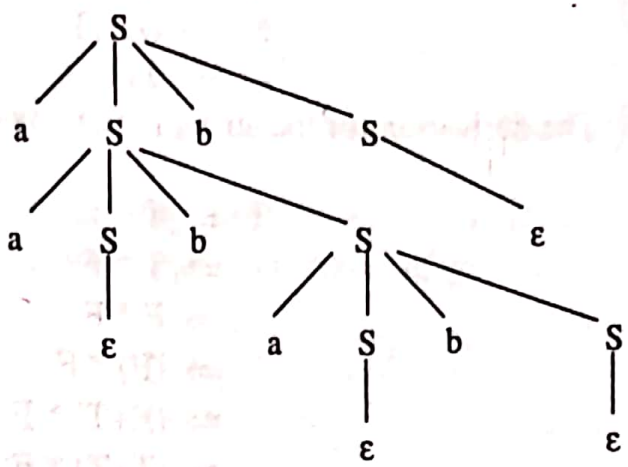
$S \Rightarrow$	$aSbS$	by using $S \rightarrow aSbS$
$\Rightarrow$	$aaSbSbS$	by using $S \rightarrow aSbS$
$\Rightarrow$	$aabSaSbSbS$	by using $S \rightarrow bSaS$
$\Rightarrow$	$aabaSbSbS$	by using $S \rightarrow \varepsilon$
$\Rightarrow$	$aababSbS$	by using $S \rightarrow \varepsilon$
$\Rightarrow$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



Scanned with CamScanner

Consider the left most derivation again for the string *aababb* but using different set of productions

$S \Rightarrow aSbS$ by using $S \rightarrow aSbS$
$\Rightarrow aaSbSbS$ by using $S \rightarrow aSbS$
$\Rightarrow aabSbS$ by using $S \rightarrow \varepsilon$
$\Rightarrow AabaSbSbS$ by using $S \rightarrow aSbS$
$\Rightarrow aababSbS$ by using $S \rightarrow \varepsilon$
$\Rightarrow aababbS$ by using $S \rightarrow \varepsilon$
$\Rightarrow aababb$ by using $S \rightarrow \varepsilon$



Since there are two parse trees for the string *aababb* by applying leftmost derivation, the grammar is ambiguous.

Note: Instead of deriving the string **aababb** in the above example, we can derive the string **"abab"** so that two different parse trees are obtained and hence we can show that the grammar is ambiguous.

Example 58: Obtain the unambiguous grammar for the grammar shown

$$E \rightarrow E+E \mid E-E$$
$$E \rightarrow E*E \mid E/E$$
$$E \rightarrow (E) \mid I$$
$$I \rightarrow a \mid b \mid c$$

and obtain the derivation for the expression (a+b) * (a-b)

It has been proved that the grammar is ambiguous. This grammar can be converted into unambiguous grammar based on the precedence. We know that identifiers such as *a*, *b* and *c* have the highest precedence, then the expression within '(' and ')' and then * or / whichever occurs first from left to right and finally + or – whichever occurs first from left to right. So, the final grammar which is unambiguous is shown below by assuming * and / have the highest priority compared to that of + and – and also by assuming all these operators are left associative.

$$I \rightarrow a \mid b \mid c$$
$$F \rightarrow (E) \mid I$$
$$T \rightarrow T*F \mid T/F \mid F$$
$$E \rightarrow E+T \mid E-T \mid T$$

So, the final grammar which is unambiguous is shown below:

Scanned with CamScanner

$$E \rightarrow E+T \mid E-T \mid T$$
$$T \rightarrow T*F \mid T/F \mid F$$
$$F \rightarrow (E) \mid I$$
$$I \rightarrow a \mid b \mid c$$

The derivation for the string (a+b) * (a-b) is shown below:

$$\begin{aligned}
E &\Rightarrow T \\
&\Rightarrow T*F \\
&\Rightarrow F*F \\
&\Rightarrow (E)*F \\
&\Rightarrow (E+T)*F \\
&\Rightarrow (T+T)*F \\
&\Rightarrow (F+T)*F \\
&\Rightarrow (I+T)*F \\
&\Rightarrow (a+T)*F \\
&\Rightarrow (a+F)*F \\
&\Rightarrow (a+I)*F \\
&\Rightarrow (a+b)*F \\
&\Rightarrow (a+b)*(E) \\
&\Rightarrow (a+b)*(E-T) \\
&\Rightarrow (a+b)*(T-T) \\
&\Rightarrow (a+b)*(F-T) \\
&\Rightarrow (a+b)*(I-T) \\
&\Rightarrow (a+b)*(a-T) \\
&\Rightarrow (a+b)*(a-F) \\
&\Rightarrow (a+b)*(a-I) \\
&\Rightarrow (a+b)*(a-b)
\end{aligned}$$

So, the string (a+b) * (a-b) is derived from the given grammar.

Note: If we assume + and – are having highest priority compared to that of * and / and all the operators left associate the grammar will be of the form

$$E \rightarrow E*T \mid E/T \mid T$$
$$T \rightarrow T+F \mid T-F \mid F$$
$$F \rightarrow (E) \mid I$$
$$I \rightarrow a \mid b \mid c$$

Note: If we assume + and – are having highest priority compared to that of * and / and all the operators right associate the grammar will be of the form

Scanned with CamScanner

$$E \rightarrow T*E \mid T/E \mid T$$
$$T \rightarrow F+T \mid F-T \mid F$$
$$F \rightarrow (E) \mid I$$
$$I \rightarrow a \mid b \mid c$$

Note: If we assume + and – are having highest priority and left associate when compared to that of * and / and if * and / operators are right associate the grammar will be of the form

$$E \rightarrow T*E \mid T/E \mid T$$
$$T \rightarrow T+F \mid T-F \mid F$$
$$F \rightarrow (E) \mid I$$
$$I \rightarrow a \mid b \mid c$$

Note: Whenever a left associative operator is involved use left recursive production and if the operator involved is right-associative, use right recursive production.

Note: Let us see "What is inherently ambiguous grammar?" A grammar G is *ambiguous* if there exists some string $w \in L(G)$ for which there are two or more distinct derivation trees. If there exists a language L for which there is no unambiguous grammar, then the language is called *inherently ambiguous language* and the grammar from which the language is derived is called ***inherently ambiguous grammar***.

For example, the grammar shown in example 58 is ambiguous. But, we have an equivalent unambiguous for the grammar (see example 58) as shown above.

There are some inherently ambiguous grammars (for these grammars unambiguous grammars do not exist). For example, consider the language

$$L = \{a^n b^n c^m d^m \mid m \geq 1, n \geq 1\} \cup \{a^n b^m c^m d^n \mid m \geq 1, n \geq 1\}$$

The grammar to generate the language is shown below:

$$S \rightarrow AB \mid C$$
$$A \rightarrow aAb \mid ab$$
$$B \rightarrow cBd \mid cd$$
$$C \rightarrow aCd \mid aDd$$
$$D \rightarrow bDc \mid bc$$

Scanned with CamScanner

The string *abcd* can be derived from the grammar as shown below:

$$\begin{aligned} S &\Rightarrow AB && \text{by using } S \rightarrow AB \\ &\Rightarrow abB && \text{by using } A \rightarrow ab \\ &\Rightarrow abcd && \text{by using } B \rightarrow cd \end{aligned}$$

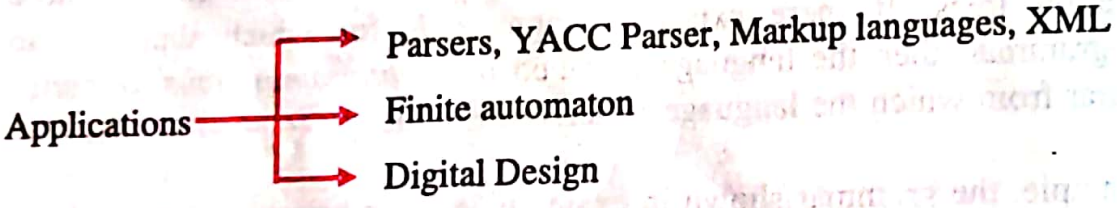
The same string *abcd* can be derived from the by applying different set of productions as shown below:

$$\begin{aligned} S &\Rightarrow C && \text{by using } S \rightarrow C \\ &\Rightarrow aDd && \text{by using } C \rightarrow aDd \\ &\Rightarrow abcd && \text{by using } D \rightarrow bc \end{aligned}$$

So, if we write the parse trees for both the derivations, the parse trees are different and naturally the grammar is ambiguous. It is not possible to obtain the unambiguous grammar for this and so it is inherently ambiguous grammar.

4.11 Applications of context free grammars

There are numerous applications of the grammars and formal languages in the field of computer science. Let us concentrate on only few applications such as



Programming Languages (Parsers, YACC Parser, Markup languages, XML)

The grammar and formal languages are very suitable to express any programming language. Building a programming language like Pascal, C, C++ etc., is very expensive and time consuming and is not the scope of the book. Just to understand that grammars are very useful in designing a programming language, let us take an example of how to find the identifiers of a particular language.

Example 59: Obtain the grammar to identify an identifier.

Note: An identifier can a variable name or a function name etc. If we define an identifier such that it should start with a letter and that letter can be followed by any combinations of letters or digits, the grammar to generate an identifier is

<identifier> → <letter><letter_digit>

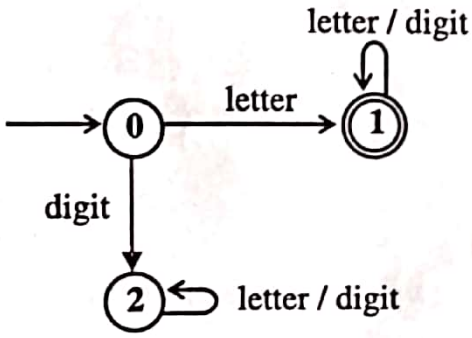
Scanned with CamScanner

<letter_digit> → <letter><letter_digit>|<digit><letter_digit>|∈
<letter> → a|b|....z|A|B|....|Z
<digit> → 0|1|2|....|9

where <identifier>, <letter_digit>, <letter>, <digit> are the variables or non-terminals and the symbols such as a, b ..z, A, B,....Z, 0, 1, 2,....9 are terminals.

Finite Automata

An automaton can be represented using a directed graph with vertices representing the states and the edges representing the transitions. The labels of the graph represent the input to be given from one state to another state. The detailed discussion of how to construct a finite automaton is given in Chapter 1 and 2. The figure below shows a finite automaton to accept only a valid identifier.



The machine initially will be in the start state 0. If the input is a *letter* it is a valid identifier and enters into state 1. In this state if the input contains a string of any combinations of *letters* or *digits*, the string will be accepted by the machine and the machine stays in state 1 accepting all digits and letters. The two concentric circles in the finite automaton indicate the final state or an accepting state. In state 0, if the first input is a *digit*, the symbol is invalid and enters into the state 2 which is the rejecting state. Once the machine enters into the rejecting state, the input string should be rejected and so the machine stays in state 2 only.

Digital Design

An automaton will be very useful in digital design because many of the circuits are based on concepts from automata theory. Even though logical implementation of a circuit is through transistors, resistors, gates, flip-flops etc., an automaton serves a bridge between its implementation and functional description of a circuit.

Scanned with CamScanner

YACC Parser-generator: The UNIX system provides a YACC command using which efficient parsers can be generated. The input to this command is a CFG but represented using different notations. In the notation used which is slightly different from that of notations used in CFG, each production is associated with an *action* which is the C code to be executed when the parse tree is created. For example, the grammar

$$E \rightarrow E+E \mid E-E$$
$$E \rightarrow E*E \mid E/E$$
$$E \rightarrow (E) \mid I$$
$$I \rightarrow a \mid b \mid c$$

using YACC notation can be written as shown below:

```
E : I          {.....}
  | E '+' E    {.....}
  | E '-' E    {.....}
  | E '*' E    {.....}
  | E '/' E    {.....}
  | E '^' E    {.....}
  | '(' E ')'  {.....}
  ;
I : 'a'        {.....}
  | 'b'        {.....}
  | 'c'        {.....}
  ;
```

Note the notations used in this representation when compared with notations used in CFG:

1. The symbol '→' is replaced by the symbol ':
2. All the productions with a given head (i.e., the non-terminals towards left of → is called a head) are grouped together and the body of the corresponding productions are separated by vertical bars and ends with terminator ';'
3. The terminals are enclosed within single quotes and the variables are not enclosed within single quotes.

Markup languages: The most familiar markup language is HTML ((Hyper Text Markup Language). HTML is used to create Hypertext documents for use on the World Wide Web. In HTML a block of text is surrounded with codes that indicate how it should appear on the computer screen. In HTML we can specify that a block of text, or a word, is linked to another file on the Internet. Hypertext Markup Language

Scanned with CamScanner

is the code used to write most documents on the World Wide Web. We can write the code using any text editor.

XML: XML stands for Extensible Markup Language. XML is a system for defining, validating, and sharing document formats. XML uses **tags** to distinguish document structures, and **attributes**. Instead of concentrating on formatting the text, XML is used to define the semantics.

YACC Parser-generator: The UNIX system provides a YACC command using which efficient parsers can be generated. The input to this command is a CFG but represented using different notations. In the notation used which is slightly different from that of notations used in CFG, each production is associated with an *action* which is the C code to be executed when the parse tree is created. For example, the grammar

$$E \rightarrow E+E \mid E-E$$
$$E \rightarrow E*E \mid E/E$$
$$E \rightarrow (E) \mid I$$
$$I \rightarrow a \mid b \mid c$$

using YACC notation can be written as shown below:

```
E : I           {.....}
  | E '+' E     {.....}
  | E '-' E     {.....}
  | E '*' E     {.....}
  | E '/' E     {.....}
  | E '^' E     {.....}
  | '(' E ')'   {.....}
  ;
I : 'a'         {.....}
  | 'b'         {.....}
  | 'c'         {.....}
  ;
```

Note the notations used in this representation when compared with notations used in CFG:

1. The symbol '→' is replaced by the symbol ':

Scanned with CamScanner

2. All the productions with a given head (i.e., the non-terminals towards left of → is called a head) are grouped together and the body of the corresponding productions are separated by vertical bars and ends with terminator ';'
3. The terminals are enclosed within single quotes and the variables are not enclosed within single quotes.

Markup languages: The most familiar markup language is HTML ((Hyper Text Markup Language). HTML is used to create Hypertext documents for use on the World Wide Web. In HTML a block of text is surrounded with codes that indicate how it should appear on the computer screen. In HTML we can specify that a block of text, or a word, is linked to another file on the Internet. Hypertext Markup Language is the code used to write most documents on the World Wide Web. We can write the code using any text editor.

XML: XML stands for Extensible Markup Language. XML is a system for defining, validating, and sharing document formats. XML uses **tags** to distinguish document structures, and **attributes**. Instead of concentrating on formatting the text, XML is used to define the semantics.

Scanned with CamScanner

Exercises:

1. Define the following terms with examples
   a. Alphabet
   b. String
   c. Concatenation of two strings
   d. star-closure
   e. Positive-closure
   f. Language
   g. Reverse of a string
   h. Length of a string
2. What is a grammar? Explain with example
3. What is derivation? Explain with example
4. Define the following terms with examples
   a. grammar
   b. sentence of a grammar
   c. sentential form
5. Obtain the grammar to generate integer
6. Let $\Sigma = \{a, b\}$. Obtain a grammar G generating set of all palindromes over Σ.
7. Obtain a grammar to generate a language of all non-palindromes over {a,b}
8. Obtain the grammar to generate the language
   $L = \{ 0^m1^m2^n \mid m \geq 1 \text{ and } n \geq 0 \}$
9. Obtain a grammar to generate the language $L = \{0^n1^{n+1} \mid n \geq 0\}$
10. Obtain the grammar to generate the languages
    a. $L = \{w \mid n_a(w) = n_b(w)\}$
    b. $L = \{w \mid n_a(w) = n_b(w)+1\}$
    c. $L = \{w \mid n_a(w) > n_b(w)\}$
    d. $L = \{w \mid n_a(w) = 2n_b(w)\}$
11. What is the language generated by the grammar
    $S \rightarrow 0A \mid \varepsilon$
    $A \rightarrow 1S$
12. Obtain a CFG to generate a string of balanced parentheses
13. Obtain a grammar to generate the language
    $L = \{ww^R \mid w \in \{a,b\}^*\}$ where w^R is reverse of w
14. Obtain a grammar to generate the language $L = \{0^n1^{2n} \mid n \geq 0\}$
15. Obtain a grammar to generate the language $L = \{0^{n+2}1^n \mid n \geq 1\}$
16. Obtain a grammar to generate the language $L = \{0^i1^j \mid i \neq j, i \geq 0 \text{ and } j \geq 0\}$
17. Obtain a grammar to generate the language
    $L = \{a^{n+2}b^m \mid n \geq 0 \text{ and } m > n\}$
18. Obtain a grammar to generate the language

Scanned with CamScanner

$L = \{a^n b^m \mid n \geq 0, m > n\}$

19. Obtain a grammar to generate the language $L = \{a^n b^{n-3} \mid n \geq 3\}$
20. Obtain a grammar to generate the language $L = L_1 L_2$ where
$L_1 = \{a^n b^m \mid n \geq 0, m > n\}$
$L_2 = \{0^n 1^{2n} \mid n \geq 0\}$
21. Obtain a grammar to generate the language $L = L_1 \cup L_2$ where
$L_1 = \{a^n b^m \mid n \geq 0, m > n\}$
$L_2 = \{0^n 1^{2n} \mid n \geq 0\}$
22. Obtain a grammar to generate the language $L = \{w : |w| \bmod 3 = 0\}$ on $\Sigma = \{a\}$
23. Obtain a grammar to generate the language $L = \{w : |w| \bmod 3 = 0\}$ on $\Sigma = \{a, b\}$
24. Obtain a grammar to generate the language $L = \{w : |w| \bmod 3 > 0\}$ on $\Sigma = \{a\}$
25. Obtain a grammar to generate the language $L = \{w : |w| \bmod 3 \neq |w| \bmod 2\}$ on $\Sigma = \{a\}$
26. Obtain a grammar to generate the language $L = \{w : |w| \bmod 3 \geq |w| \bmod 2\}$ on $\Sigma = \{a\}$
27. Obtain a grammar to generate the set of all strings with exactly one *a* when $\Sigma = \{a, b\}$
28. Obtain a grammar to generate the set of all strings with at least one *a* when $\Sigma = \{a, b\}$
29. Obtain a grammar to generate the set of all strings with no more than three *a*'s when $\Sigma = \{a, b\}$
30. Name some of the applications of formal languages and grammar
31. Obtain a grammar to grammar to identify an identifier
32. How an automaton can be used to add two binary numbers?
33. What is leftmost derivation? Explain with example.
34. What is rightmost derivation? Explain with example.
35. What is a derivation tree (or parse tree)? Explain with example
36. What is the *yield* of a tree? Explain with example
37. What is partial parse tree (or partial derivation tree)? Explain with example.
38. What is an ambiguous grammar?

Scanned with CamScanner

39. Is the following grammar ambiguous?
    S → iCtS | iCtSeS | a
    C → b
40. Is the grammar ambiguous?
    S → AB | aaB
    A → a | Aa
    B → b
41. Show that the following grammar is ambiguous
    S → aSbS
    S → bSaS
    S → ε
42. Obtain the unambiguous grammar for the grammar shown
    E → E + E | E - E
    E → E * E | E / E
    E → (E) | I
    I → a | b | c
    and obtain the derivation for the expression (a+b) * (a-b)
43. What is inherently ambiguous grammar?
44. What is an S-Grammar (Simple Grammar)? Explain with example.
45. Find a Simple Grammar (S-Grammar) for the regular expression aaa*b + b
46. Find a Simple Grammar (S-Grammar) to generate the language $L = \{a^n b^n \mid n \geq 1\}$
47. In what way BNF notations are different from the usual representation of the grammar? Give an example.
48. Use BNF notation and describe the while statement in C language. Assume that assignment statement and condition for while are defined already.
49. Give the BNF notation to write a C program. Provide 6 or 7 productions generally describing the main program. Assume the rest are defined.
50. What language is accepted by the following grammars?
    a. S → 0S0 | 1S1 | ε
    b. S → 0S0 | 1S1 | 0 | 1
    c. S → 0S1 | 1S1 | ε
       A → 0B1 | 1B0
       B → 0B0 | 1B1 | 0 | 1 | ε
    d. S → cS | cSdS | ε

Scanned with CamScanner

e. $S \rightarrow cS \mid dS \mid c$
f. $S \rightarrow SS \mid dS \mid Sd \mid c$

51. Is the grammar $S \rightarrow SS \mid (S) \mid \varepsilon$ ambiguous? (Ans : Yes) obtain two derivations by applying LMD or obtain two derivations by applying RMD.

52. What are the limitations of regular languages?
53. What is a context free grammar? Explain with example.
54. Let G = (V,T,P,S) be a CFG where

$V = \{ S \}$
$T = \{a, b\}$
$P = \{$
$S \rightarrow aSa \mid bSb \mid \varepsilon$
$\}$
S is the start symbol.
What is the language generated by this grammar?

55. Show that the language $L = \{ a^m b^n \mid m \neq n \}$ is context free.
56. Draw a CFG to generate a language consisting of equal number of a's and b's
57. Draw a CFG on {a, b} to generate a language $L = \{ a^n w w^R b^n \mid w \in \Sigma^*, n \geq 1\}$
58. Obtain a context free grammar to generate properly nested parentheses structures involving three kinds of parentheses ( ), [ ] and { }.
59. Obtain a context free grammar to generate the following language
$L = \{w \mid w \in \{a, b\}^*, n_a(v) \geq n_b(v)$ where v is any prefix of w$\}$
60. Obtain a context free grammar to generate the following language
$L = \{01(1100)^n 110(10)^n \mid n \geq 0\}$
61. Is the following language Context free?
$L = \{a^n b^n \mid n \geq 0\}$
62. Obtain a context free grammar to generate the following language
$L = \{a^n b^m : m > n$ and $n \geq 0\}$
63. Obtain a CFG to generate unequal number of a's and b's
64. For the regular expression (011+1)*(01)* obtain the context free grammar.
65. What is leftmost derivation? Explain with example.
66. What is rightmost derivation? Explain with example.
67. What is a derivation tree (or parse tree)? Explain with example
68. What is the *yield* of a tree? Explain with example
69. What is partial parse tree (or partial derivation tree)? Explain with example.
70. What is an ambiguous grammar?

Scanned with CamScanner

71. Consider the grammar shown below from which any arithmetic expression can be obtained.

$$E \rightarrow E+E$$
$$E \rightarrow E-E$$
$$E \rightarrow E*E$$
$$E \rightarrow E/E$$
$$E \rightarrow (E) \mid id$$

Show that the grammar is ambiguous.

72. Is the following grammar ambiguous?

$$S \rightarrow aS \mid X$$
$$X \rightarrow aX \mid a$$

73. Is the following grammar ambiguous?

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

74. Is the grammar ambiguous?

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

75. Show that the following grammar is ambiguous

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

76. Obtain the unambiguous grammar for the grammar shown

$$E \rightarrow E+E \mid E-E$$
$$E \rightarrow E*E \mid E/E$$
$$E \rightarrow (E) \mid I$$
$$I \rightarrow a \mid b \mid c$$

and obtain the derivation for the expression (a+b) * (a-b)

77. What is inherently ambiguous grammar?

78. What is an S-Grammar (Simple Grammar)? Explain with example.

79. Find a Simple Grammar (S-Grammar) for the regular expression $aaa^*b + b$

80. Find a Simple Grammar (S-Grammar) to generate the language $L = \{a^n b^n \mid n \geq 1\}$

Scanned with CamScanner

81. In what way BNF notations are different from the usual representation of the grammar? Give an example.
82. Use BNF notation and describe the while statement in C language. Assume that assignment statement and condition for while are defined already.
83. Give the BNF notation to write a C program. Provide 6 or 7 productions generally describing the main program. Assume the rest are defined.
84. Obtain a derivation tree for the string a + b * a + b from the grammar shown below:

$$
\begin{aligned}
E &\rightarrow E+E \\
E &\rightarrow E-E \\
E &\rightarrow E*E \\
E &\rightarrow E/E \\
E &\rightarrow a \mid b
\end{aligned}
$$

85. Consider the grammar :

$$
\begin{aligned}
S &\rightarrow 0 \mid 01S1 \mid 0A1 \\
A &\rightarrow 1S \mid 0AA1
\end{aligned}
$$

Is the grammar ambiguous? Ans: yes

86. What language is accepted by the following grammars?
    g. $S \rightarrow 0S0 \mid 1S1 \mid \varepsilon$
    h. $S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1$
    i. $S \rightarrow 0S1 \mid 1S1 \mid \varepsilon$
       $A \rightarrow 0B1 \mid 1B0$
       $B \rightarrow 0B0 \mid 1B1 \mid 0 \mid 1 \mid \varepsilon$
    j. $S \rightarrow cS \mid cSdS \mid \varepsilon$
    k. $S \rightarrow cS \mid dS \mid c$
    l. $S \rightarrow SS \mid dS \mid Sd \mid c$

87. Is the grammar $S \rightarrow SS \mid (S) \mid \varepsilon$ ambiguous? (Ans : Yes) obtain two derivations by applying LMD or obtain two derivations by applying RMD.

Scanned with CamScanner

6.3 Simplification of CFG

In a CFG, it may be necessary to eliminate some of the useless symbols and productions. Let G = (V, T, P, S) be a CFG. In the grammar G, some of the symbols or productions may not be used to derive a string and some symbols and productions may not be reachable from the start symbol. So, these symbols and productions which are not used in any sentential from are useless and the corresponding productions can be eliminated. For example, consider the grammar

$$S \rightarrow aA \mid B$$
$$A \rightarrow aA \mid a$$

In this grammar if we apply the production S → B, from B, a string can never be derived. So, the symbol B and the production S → B are useless and can be eliminated. So, in this section, let us concentrate on how a grammar can be simplified by eliminating useless symbols and variables.

Scanned with CamScanner

Example 6.4: **Eliminate the useless symbols in the grammar**

$S \rightarrow aA \mid bB$
$A \rightarrow aA \mid a$
$B \rightarrow bB$
$D \rightarrow ab \mid Ea$
$E \rightarrow aC \mid d$

Stage 1 : Applying the algorithm shown in stage1, we can obtain a set of variables from which we get only string of terminals and is shown below.

ov	nv	Productions
ϕ	A, D, E	$A \rightarrow a$ $D \rightarrow ab$ $E \rightarrow d$
A,D,E	A,D,E,S	$S \rightarrow aA$ $A \rightarrow aA$ $D \rightarrow Ea$
A,D,E,S	A,D,E,S	-

The resulting grammar $G_1 = (V_1, T_1, P_1, S)$ where

$V_1 = \{A,D,E,S\}$
$T_1 = \{a, b, d\}$
$P_1 = \{$
$A \rightarrow a \mid aA$
$D \rightarrow ab \mid Ea$
$E \rightarrow d$
$S \rightarrow aA$
$\}$
S is the start symbol

contains all those variables in V_1 such that $A \overset{+}{\Rightarrow} w$ where $w \in T^+$.

Stage 2: Applying the algorithm given in stage 2 of the theorem 6.2, we obtain the symbols such that each symbol X is reachable from the start symbol S as shown below.

P^1	T^1	V^1
-	-	S
$S \rightarrow aA$	a	S, A
$A \rightarrow a \mid aA$	a	S, A

Scanned with CamScanner

The resulting grammar $G^1 = (V^1, T^1, P^1, S)$ where

V^1 = { S, A }
T^1 = {a}
P^1 = {
S → aA
A → a | aA
}
S is the start symbol

such that each symbol X in $(V^1 \cup T^1)$ has a derivation of the form

$$S \overset{*}{\Rightarrow} \alpha X \beta \overset{*}{\Rightarrow} w.$$

Example 6.5: Simplify the following grammar

S → aA | a | Bb | cC
A → aB
B → a | Aa
C → cCD
D → ddd

Stage 1 : Applying the algorithm shown in stage1 of theorem 6.2, we can obtain a set of variables from which we get only string of terminals and is shown below.

ov	nv	Productions
ϕ	S, B, D	S → a B → a D → ddd
S, B, D	S, B, D, A	S → Bb A → aB
S, B, D, A	S, B, D, A	S → aA B → Aa

The resulting grammar $G_1 = (V_1, T_1, P_1, S)$ where

V_1 = {S, B, D, A}
T_1 = {a, b, d}

P_1 = {
S → a | Bb | aA
B → a | Aa

Scanned with CamScanner

D → ddd
A → aB
}
S is the start symbol

contains all those variables in V_1 such that $A \overset{+}{\Rightarrow} w$.

Stage 2: Applying the algorithm given in stage 2 of the theorem 6.2, we obtain the symbols such that each symbol X is reachable from the start symbol S as shown below.

P^1	T^1	V^1
-	-	S
S → a | Bb | Aa	a, b	S, A, B
A → aB	a, b	S, A, B
B → a | Aa	a, b,	S, A, B

The resulting grammar $G^1 = (V^1, T^1, P^1, S)$ where

V^1 = { S, A, B }
T^1 = {a, b}
P^1 = {
S → a | Bb | aA
A → aB
B → a | Aa
}
S is the start symbol

such that each symbol X in $(V^1 \cup T^1)$ has a derivation of the form

$$S \overset{*}{\Rightarrow} \alpha X \beta \overset{*}{\Rightarrow} w.$$

6.4 Eliminating ε - productions

A production of the form A → ε is undesirable in a CFG, unless an empty string is derived from the start symbol. Suppose, the language generated from a grammar G does not derive any empty string and the grammar consists of ε-productions. Such ε-productions can be removed. An ε-production is defined as follows:

Definition: Let G = (V, T, P, S) be a CFG. A production in P of the form

$$A \rightarrow \varepsilon$$

Scanned with CamScanner

is called an ε-production or NULL production. After applying the production the variable A is erased. For each A in V, if there is a derivation of the form

$$A \overset{*}{\Rightarrow} \varepsilon$$

then A is a nullable variable.

Example 6.6: **Consider the grammar**

$$\begin{aligned} S &\rightarrow ABCa \mid bD \\ A &\rightarrow BC \mid b \\ B &\rightarrow b \mid \varepsilon \\ C &\rightarrow c \mid \varepsilon \\ D &\rightarrow d \end{aligned}$$

In this grammar, the productions

$$B \rightarrow \varepsilon$$
$$C \rightarrow \varepsilon$$

are ε- productions and the variables B, C are nullable variables. Because there is a production

$$A \rightarrow BC$$

and both B and C are nullable variables, then A is also a nullable variable.

Definition: Let G = (V, T, P, S) be a CFG where V set of variables, T is set of terminals, P is set of productions and S is the start symbol. A nullable variable is defined as follows.

1. If $A \rightarrow \varepsilon$ is a production in P, then A is a nullable variable.
2. If $A \rightarrow B_1B_2 \ldots B_n$ is a production in P, and if $B_1, B_2, \ldots B_n$ are nullable variables, then A is also a nullable variable
3. The variables for which there are productions of the form shown in step 1 and step2 are nullable variables.

Scanned with CamScanner

Even though a grammar G has some ε-productions, the language may not derive a language containing empty string. So, in such cases, the ε- productions or NULL productions are not needed and they can be eliminated.

Scanned with CamScanner

Even though a grammar G has some ε-productions, the language may not derive a language containing empty string. So, in such cases, the ε- productions or NULL productions are not needed and they can be eliminated.

Theorem 6.3: Let G = (V,T,P,S) where $L(G) \neq \varepsilon$. We can effectively find an equivalent grammar G^1 with no $\in$ - productions such that $L(G^1) = L(G) - \varepsilon$.

Proof : The grammar G^1 can be obtained from G in two steps.

Step1: find the set of nullable variables in the grammar G using the following algorithm.

```
ov = ϕ

nv = { A | A → ε }

while ( ov != nv )
{
        ov = nv
        nv = ov ∪ { A | A → α and α ∈ ov* }
}

V = ov
```

Once the control comes out of the while loop, the set V contains only the nullable variables.

Step2: Construction of productions P^1. Consider a production of the form

$$A \rightarrow X_1X_2X_3.....X_n \text{ , } n \geq 1$$

where each X_i is in (V U T). In a production, take all possible combinations of nullable variables and replace the nullable variables with ε one by one and add the resulting productions to P^1. If the given production is not an ε- production, add it to P^1.

Suppose, A and B are nullable variables in the production, then

1. First add the production to P^1.

Scanned with CamScanner

2. Replace A with ε in the given production and add the resulting production to P^1.
3. Replace B with ε in the given production and add the resulting production to P^1.
4. Replace A and B with ε and add the resulting production to P^1.
5. If all symbols on right side of production are nullable variables, the resulting production is an ε- production and do not add this to P^1.

Thus, the resulting grammar G^1 obtained, generates the same language as generated by G without ε and the proof is straight forward.

Example 6.7: Eliminate all ε- productions from the grammar

$$\begin{aligned} S &\rightarrow ABCa \mid bD \\ A &\rightarrow BC \mid b \\ B &\rightarrow b \mid \varepsilon \\ C &\rightarrow c \mid \varepsilon \\ D &\rightarrow d \end{aligned}$$

Step1: Obtain the set of nullable variables from the grammar. This can be done using step 1 of theorem 6.3 as shown below.

ov	nv	Productions
ϕ	B,C	B → ε C → ε
B,C	B,C,A	A → BC
B,C,A	B,C,A	-

V = {B,C,A} are all nullable variables.

Step2: Construction of productions P^1.

Productions	Resulting productions (P^1)
S → ABCa	S → ABCa | BCa | ACa | ABa|Ca | Aa | Ba | a
S → bD	S → bD
A → BC | b	A → BC | B | C | b
B → b | ε	B → b
C → c | ε	C → c
D → d	D → d

Scanned with CamScanner

The grammar $G^1 = (V^1, T^1, P^1, S)$ where

V^1 = {S, A, B, C, D}
T^1 = {a, b, c, d }
P^1 = {

S → ABCa | BCa | ACa | ABa | Ca | Aa | Ba | a | bD
A → BC | B | C | b
B → b
C → c
D → d

}

S is the start symbol

Example 6.8: Eliminate all ε- productions from the grammar

S → BAAB
A → 0A2 | 2A0 | ε
B → AB | 1B | ε

Step1: Obtain the set of nullable variables from the grammar. This can be achieved using step 1 of theorem 6.3 as shown below.

ov	nv	Productions
ϕ	A,B	A → ε B → ε
A, B	A, B, S	S → BAAB
A, B, S	A, B, S	-

V = {S, A, B} are all nullable variables.

Step2: Construction of productions P^1. Add a non ∈-production in P to P^1. Take all the combinations of nullable variables in a production, delete subset of nullable variables one by one and add the resulting productions to P^1.

Productions	Resulting productions (P^1)
S → BAAB	S → BAAB | AAB | BAB | BAA | AB | BB | BA | AA | A | B
A → 0A2	A → 0A2 | 02
A → 2A0	A → 2A0 | 20
B → AB	B → AB | B | A
B → 1B	B → 1B | 1

Scanned with CamScanner

We can delete the productions of the form $A \rightarrow A$. In P^1, the production $B \rightarrow B$ can be deleted and the final grammar obtained after eliminating ε-productions is shown below.

The grammar $G^1 = (V^1, T^1, P^1, S)$ where

V^1 = {S, A, B}
T^1 = {0, 1, 2 }
P^1 = {

S → BAAB | AAB | BAB | BAA | AB | BB | BA | AA | A | B
A → 0A2 | 02 | 2A0 | 20
B → AB | A | 1B | 1

}
S is the start symbol

6.5 Eliminating unit productions

Consider the productions $A \rightarrow B$. The left hand side of the production and right hand side of the production contains only one variable. Such productions are called unit productions. Formally, a unit production is defined as follows.

Definition: Let G = (V, T, P, S) be a CFG. Any production in G of the form

$$A \rightarrow B$$

where $A, B \in V$ is a unit-production.

In any grammar, the unit productions are undesirable. This is because one variable is simply replaced by another variable. Consider the productions

$A \rightarrow B$
$B \rightarrow aB \mid b$

In this example,

$B \rightarrow aB$
$B \rightarrow b$

are non unit productions. Since B is generated from A, whatever is generated by B, the same things can be generated from A also. So, we can have

$A \rightarrow aB$
$B \rightarrow b$

and the production $A \rightarrow B$ can be deleted.

Scanned with CamScanner

Theorem 6.4: Let G = (V, T, P, S) be a CFG and has unit productions and no ∈-productions. An equivalent grammar G_1 without unit productions can be obtained such that $L(G) = L(G_1)$ i.e., any language generated by G is also generated by G_1. But, the grammar G_1 has no unit productions.

A unit production in grammar G can be eliminated using the following steps:

1. Remove all the productions of the form $A \rightarrow A$
2. Add all non unit productions to P_1.
3. For each variable A find all variables B such that

$$A \overset{*}{\Rightarrow} B$$

i.e., in the derivation process from A, if we encounter only one variable in a sentential form say B (no terminals should be there), obtain all such variables.

4. Obtain a dependency graph. For example, if we have the productions

$$A \rightarrow B$$
$$B \rightarrow C$$
$$C \rightarrow B$$

the dependency graph will be of the form

5. Note from the dependency graph that
   a. $A \overset{*}{\Rightarrow} B$ i.e., B can be obtained from A
      So, all non-unit productions generated from B can also be generated from A
   b. $A \overset{*}{\Rightarrow} C$ i.e., C can be obtained from A
      So, all non-unit productions generated from C can also be generated from A
   c. $B \overset{*}{\Rightarrow} C$ i.e., C can be obtained from B
      So, all non-unit productions generated from C can also be generated from B
   d. $C \overset{*}{\Rightarrow} B$ i.e., B can be obtained from C
      So, all non-unit productions generated from B can also be generated from C
6. Finally, the unit productions can be deleted from the grammar G.
7. The resulting grammar G_1, generates the same language as accepted by G.

Scanned with CamScanner

Example 6.9: **Eliminate all unit productions from the grammar**

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow a \\ B &\rightarrow C \mid b \\ C &\rightarrow D \\ D &\rightarrow E \mid bC \\ E &\rightarrow d \mid Ab \end{aligned}$$

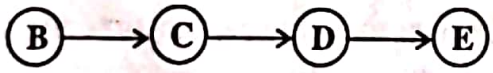
The non unit productions of the grammar G are shown below:

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow a \\ B &\rightarrow b \\ D &\rightarrow bC \\ E &\rightarrow d \mid Ab \end{aligned} \qquad (6.1)$$

The unit productions of the grammar G are shown below:

$$\begin{aligned} B &\rightarrow C \\ C &\rightarrow D \\ D &\rightarrow E \end{aligned}$$

The dependency graph for the unit-productions is shown below:



It is clear from the dependency graph that $D \overset{*}{\Rightarrow} E$. So, all non unit productions generated from E can also be generated from D. The non unit productions from E are

$$E \rightarrow d \mid Ab \qquad (6.2)$$

can also be obtained from D.

$$D \rightarrow d \mid Ab$$

The resulting D productions are

$$\begin{aligned} D &\rightarrow bC \quad \text{(from 6.1)} \\ D &\rightarrow d \mid Ab \end{aligned} \qquad (6.3)$$

Scanned with CamScanner

From the dependency graph it is clear that, $C \overset{*}{\Rightarrow} E$. So, the non unit productions from E shown in (6.2) can be generated from C. Therefore,

$$C \rightarrow d \mid Ab$$

From the dependency graph it is clear that, $C \overset{*}{\Rightarrow} D$. So, the non unit productions from D shown in (6.3) can be generated from C. Therefore,

$$C \rightarrow bC$$
$$C \rightarrow d \mid Ab \qquad (6.4)$$

From the dependency graph it is clear that $B \overset{*}{\Rightarrow} C$, $B \overset{*}{\Rightarrow} D$, $D \overset{*}{\Rightarrow} E$. So, all the productions obtained from B can be obtained using (6.1), (6.2), (6.3) and (6.4) and the resulting productions are:

$$B \rightarrow b$$
$$B \rightarrow d \mid Ab$$
$$B \rightarrow bC \qquad (6.5)$$

The final grammar obtained after eliminating unit productions can be obtained by combining the productions (6.1), (6.2), (6.3), (6.4) and (6.5) and is shown below:

V^1 = {S, A, B, C, D, E}
T^1 = {a, b, d}
P^1 = {

S → AB
A → a
B → b | d | Ab | bC
C → bC | d | Ab
D → bC | d | Ab
E → d | Ab

}
S is the start symbol

Example 6.10: Eliminate unit productions from the grammar

$$S \rightarrow A0 \mid B$$
$$B \rightarrow A \mid 11$$
$$A \rightarrow 0 \mid 12 \mid B$$

The dependency graph for the unit productions

$$S \rightarrow B$$
$$B \rightarrow A$$
$$A \rightarrow B$$

Scanned with CamScanner

is shown below.

The non unit productions are

$$\begin{aligned} S &\rightarrow A0 \\ B &\rightarrow 11 \\ A &\rightarrow 0 \mid 12 \end{aligned} \qquad (6.6)$$

It is clear from the dependency graph that $S \overset{*}{\Rightarrow} B$, $S \overset{*}{\Rightarrow} A$, $B \overset{*}{\Rightarrow} A$ and $A \overset{*}{\Rightarrow} B$. So, the new productions from S, A and B are

$$\begin{aligned} S &\rightarrow 11 \mid 0 \mid 12 \\ B &\rightarrow 0 \mid 12 \\ A &\rightarrow 11 \end{aligned} \qquad (6.7)$$

The resulting grammar without unit productions can be obtained by combining (6.6) and (6.7) and is shown below:

$V^1 = \{S, A, B\}$
$T^1 = \{0, 1, 2\}$
$P^1 = \{$
$S \rightarrow A0 \mid 11 \mid 0 \mid 12$
$A \rightarrow 0 \mid 12 \mid 11$
$B \rightarrow 11 \mid 0 \mid 12$
$\}$
S is the start symbol

Example 6.11: Eliminate unit productions from the grammar

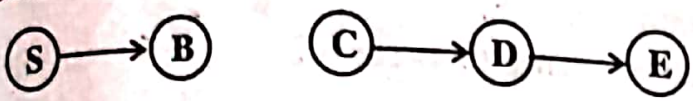
$$\begin{aligned} S &\rightarrow Aa \mid B \mid Ca \\ B &\rightarrow aB \mid b \\ C &\rightarrow Db \mid D \\ D &\rightarrow E \mid d \\ E &\rightarrow ab \end{aligned}$$

The dependency graph for the unit productions

Scanned with CamScanner

$S \rightarrow B$
$C \rightarrow D$
$D \rightarrow E$

is shown below.



The non unit productions are

$S \rightarrow Aa \mid Ca$
$B \rightarrow aB \mid b$
$C \rightarrow Db$
$D \rightarrow d$
$E \rightarrow ab$

It is clear from the first dependency graph that $S \overset{*}{\Rightarrow} B$ and so whatever is derivable from B it is also derivable from S and the resulting S-productions are: (6.8)

$S \rightarrow aB \mid b$

It is clear from the second dependency graph $C \overset{*}{\Rightarrow} D$ and $D \overset{*}{\Rightarrow} E$. So, whatever is derivable from E is also derivable from D and the resulting D-productions are: (6.9)

$D \rightarrow ab \mid d$

and the D productions are also derivable from C since $C \overset{*}{\Rightarrow} D$. So, the resulting C-productions are: (6.10)

$C \rightarrow Db \mid ab \mid d$

The resulting grammar without unit productions can be obtained by combining (6.8), (6.9), (6.10) and (6.11) and is shown below: (6.11)

$V^1 = \{S, A, B, C, D, E\}$
$T^1 = \{a, b, d\}$
$P^1 = \{$
$S \rightarrow Aa \mid Ca \mid aB \mid b$
$B \rightarrow aB \mid b$
$C \rightarrow Db \mid ab \mid d$
$D \rightarrow d \mid ab$
$E \rightarrow ab$
$\}$
S is the start symbol

Scanned with CamScanner

Note : Given any grammar, all undesirable productions can be eliminated by removing

1. ε-productions using theorem 6.3
2. unit productions using theorem 6.4
3. useless symbols and productions using theorem 6.2

in sequence. The final grammar obtained does not have any undesirable productions.

6.6 Chomsky Normal Form

In a CFG, there is no restriction on the right hand side of a production. The restrictions are imposed on the right hand side of productions in a CFG resulting in various normal forms. The different normal forms that we discuss are:

1. Chomsky Normal Form (CNF)
2. Greiback Normal Form (GNF)

Chomsky normal form can be defined as follows.

Definition: Let G = (V, T, P, S) be a CFG. The grammar G is said to be in CNF if all productions are of the form

$$A \rightarrow BC$$

or

$$A \rightarrow a$$

where A, B and C $\in$ V and a $\in$ T.

Note that if a grammar is in CNF, the right hand side of the production should contain two symbols or one symbol. If there are two symbols on the right hand side those two symbols must be non-terminals and if there is only one symbol, that symbol must be a terminal.

Theorem 6.5: Let G = (V, T, P, S) be a CFG which generates context free language without ε. We can find an equivalent context free grammar $G^1 = (V^1, T, P^1, S)$ in CNF such that

$$L(G) = L(G^1)$$

Scanned with CamScanner

i.e., all productions in G^1 are of the form

$$A \rightarrow BC$$

or

$$A \rightarrow a.$$

Proof: Let the grammar G has no ε-productions and unit productions. The grammar G^1 can be obtained using the following steps.

Step1: Consider the productions of the form

$$A \rightarrow X_1X_2X_3\text{- - - -}X_n$$

where $n \geq 2$ and each $X_i \in (V \cup T)$ i.e., consider the productions having more than two symbols on the right hand side of the production. If X is a terminal say *a*, then replace this terminal by a corresponding non- terminal B_a and introduce the production

$$B_a \rightarrow a$$

The non-terminals on the right hand side of the production are retained. The resulting productions are added to P_1. The resulting context free grammar $G_1 = (V_1, T, P_1, S)$ where each production in P_1 is of the form

$$A \rightarrow A_1A_2\text{-----}A_n$$

or

$$A \rightarrow a$$

generates the same language as accepted by grammar G. So, $L(G) = L(G_1)$.

Step2: Restrict the number of variables on the right hand side of the production. Add all the productions of G_1 which are in CNF to P^1. Consider a production of the form

$$A \rightarrow A_1A_2\text{-------}A_n$$

where $n \geq 3$ (Note that if $n = 2$, the production is already in CNF and *n* can not be equal to 1. Because if $n = 1$, there is only one symbol and it is a terminal which again is in CNF). The A-production can be written as

Scanned with CamScanner

$$
\begin{aligned}
A &\rightarrow A_1 D_1 \\
D_1 &\rightarrow A_2 D_2 \\
D_3 &\rightarrow A_3 D_3 \\
&\vdots \\
D_{n-2} &\rightarrow A_{n-1} A_n
\end{aligned}
$$

These productions are added to P^1 and new variables are added to V^1. The grammar thus obtained is in CNF. The resulting grammar $G^1 = (V^1, T, P^1, S)$ generates the same language as accepted by G i.e. $L(G) = L(G^1)$.

Example 6.12: Consider the grammar

$$
\begin{aligned}
S &\rightarrow 0A \mid 1B \\
A &\rightarrow 0AA \mid 1S \mid 1 \\
B &\rightarrow 1BB \mid 0S \mid 0
\end{aligned}
$$

Obtain the grammar in CNF.

All productions which are in CNF are added to P_1. The productions which are in standard form and added to P_1 are:

$$
\begin{aligned}
A &\rightarrow 1 \\
B &\rightarrow 0
\end{aligned} \qquad (6.8)
$$

Consider the productions, which are not in CNF. Replace the terminal *a* on right hand side of the production by a non-terminal A and introduce the production $A \rightarrow a$. This step has to be carried out for each production which are not in CNF.

The table below shows the action taken indicating which terminal is replaced by the corresponding non-terminal and what is the new production introduced. The last column shows the resulting productions.

Given Productions	Action	Resulting productions
$S \rightarrow 0A \mid 1B$	Replace 0 by B_0 and introduce the production $B_0 \rightarrow 0$ Replace 1 by B_1 and introduce the production $B_1 \rightarrow 1$	$S \rightarrow B_0A \mid B_1B$ $B_0 \rightarrow 0$ $B_1 \rightarrow 1$

Scanned with CamScanner

$A \rightarrow 0AA/1S$	Replace 0 by B_0 and introduce the production $B_0 \rightarrow 0$ Replace 1 by B_1 and introduce the production $B_1 \rightarrow 1$	$A \rightarrow B_0AA/B_1S$ $B_0 \rightarrow 0$ $B_1 \rightarrow 1$
$B \rightarrow 1BB/0S$	Replace 0 by B_0 and introduce the production $B_0 \rightarrow 0$ Replace 1 by B_1 and introduce the production $B_1 \rightarrow 1$	$B \rightarrow B_1BB/B_0S$ $B_1 \rightarrow 1$ $B_0 \rightarrow 0$

The grammar $G_1 = (V_1, T, P_1, S)$ can be obtained by combining the productions obtained from the last column in the table and the productions shown in (6.8).

$V_1 = \{S, A, B, B_0, B_1\}$

$T_1 = \{0, 1\}$

$P_1 = \{$

$S \rightarrow B_0A \mid B_1B$

$A \rightarrow B_0AA \mid B_1S \mid 1$

$B \rightarrow B_1BB \mid B_0S \mid 0$

$B_0 \rightarrow 0$

$B_1 \rightarrow 1$

$\}$

S is the start symbol

Step2: Restricting the number of variables on the right hand side of the production to 2. The productions obtained after step1 are:

$S \rightarrow B_0A \mid B_1B$

$A \rightarrow B_0AA \mid B_1S \mid 1$

$B \rightarrow B_1BB \mid B_0S \mid 0$

$B_0 \rightarrow 0$

$B_1 \rightarrow 1$

In the above productions, the productions which are in CNF are

$S \rightarrow B_0A \mid B_1B$

$A \rightarrow B_1S \mid 1$

$B \rightarrow B_0S \mid 0$

$B_0 \rightarrow 0$

$B_1 \rightarrow 1$

(6.9)

and add these productions to P^1. The productions which are not in CNF are

Scanned with CamScanner

$$A \rightarrow B_0AA$$
$$B \rightarrow B_1BB$$

The following table shows how these productions are changed to CNF so that only two variables are present on the right hand side of the production.

Given Productions	Action	Resulting productions
$A \rightarrow B_0AA$	Replace AA on R.H.S with variable D_1 and introduce the production $D_1 \rightarrow AA$	$A \rightarrow B_0D_1$ $D_1 \rightarrow AA$
$B \rightarrow B_1BB$	Replace BB on R.H.S with variable D_2 and introduce the production $D_2 \rightarrow BB$	$B \rightarrow B_1D_2$ $D_2 \rightarrow BB$

(6.10)

The final grammar which is in CNF can be obtained by combining the productions in (6.9) and (6.10). The grammar $G^1 = (V^1, T, P^1, S)$ is in CNF where

$V_1 = \{S, A, B, B_0, B_1, D_1, D_2\}$

$T_1 = \{0, 1\}$

$P_1 = \{$

$S \rightarrow B_0A \mid B_1B$

$A \rightarrow B_1S \mid 1 \mid B_0D_1$

$B \rightarrow B_0S \mid 0 \mid B_1D_2$

$B_0 \rightarrow 0$

$B_1 \rightarrow 1$

$D_1 \rightarrow AA$

$D_2 \rightarrow BB$

$\}$

S is the start symbol

Scanned with CamScanner

7.1.1 Eliminating Useless Symbols

We say a symbol X is *useful* for a grammar $G = (V, T, P, S)$ if there is some derivation of the form $S \stackrel{*}{\Rightarrow} \alpha X \beta \stackrel{*}{\Rightarrow} w$, where w is in T^*. Note that X may be in either V or T, and the sentential form $\alpha X \beta$ might be the first or last in the derivation. If X is not useful, we say it is *useless*. Evidently, omitting useless symbols from a grammar will not change the language generated, so we may as well detect and eliminate all useless symbols.

Our approach to eliminating useless symbols begins by identifying the two things a symbol has to be able to do to be useful:

1. We say X is *generating* if $X \stackrel{*}{\Rightarrow} w$ for some terminal string w. Note that every terminal is generating, since w can be that terminal itself, which is derived by zero steps.

2. We say X is *reachable* if there is a derivation $S \stackrel{*}{\Rightarrow} \alpha X \beta$ for some α and β.

Surely a symbol that is useful will be both generating and reachable. If we eliminate the symbols that are not generating first, and then eliminate from the remaining grammar those symbols that are not reachable, we shall, as will be proved, have only the useful symbols left.

Example 7.1: Consider the grammar

$$\begin{aligned} S &\to aAa \mid aBC \\ A &\to aS \mid bD \\ B &\to aBa \mid b \\ C &\to abb \mid DD \\ D &\to aDa \end{aligned}$$

Initially we find B and C to be the only nonterminal symbols which directly generate a terminal string: hence it is generating. S is generating because of the rule $S \to aA$ and A is generating because of the rule $A \to aS$. Thus the only symbol which is not generating is D. Eliminating D and those productions involving D we get the grammar:

$$\begin{aligned} S &\to aAa \mid aBC \\ A &\to aS \\ B &\to aBa \mid b \\ C &\to abb \end{aligned}$$

Now we find that only S, A, B, a and b are reachable from S. Eliminating C and all productions involving it we get

$$\begin{aligned} S &\to aAa \\ A &\to aS \\ B &\to aBa \mid b \end{aligned}$$

Scanned with CamScanner

Note that if we check for reachability first we find that all symbols of the grammar

$$\begin{aligned} S &\to aAa \mid aBC \\ A &\to aS \mid bD \\ B &\to aBa \mid b \\ C &\to abb \mid DD \\ D &\to aDa \end{aligned}$$

are reachable. If we now eliminate D since it is not generating we are left with a grammar which still has a useless symbol C. □

Theorem 7.2: Let $G = (V, T, P, S)$ be a CFG, and assume that $L(G) \neq \emptyset$; i.e., G generates at least one string. Let $G_1 = (V_1, T_1, P_1, S)$ be the grammar we obtain by the following steps:

1. First eliminate nongenerating symbols and all productions involving one or more of those symbols. Let $G_2 = (V_2, T_2, P_2, S)$ be this new grammar. Note that S must be generating, since we assume $L(G)$ has at least one string, so S has not been eliminated.

2. Second, eliminate all symbols that are not reachable in the grammar G_2.

Then G_1 has no useless symbols, and $L(G_1) = L(G)$.

PROOF: Suppose X is a symbol that remains; i.e., X is in $V_1 \cup T_1$. We know that $X \overset{*}{\underset{G}{\Rightarrow}} w$ for some w in T^*. Moreover, every symbol used in the derivation of w from X is also generating. Thus, $X \overset{*}{\underset{G_2}{\Rightarrow}} w$.

Since X was not eliminated in the second step, we also know that there are α and β such that $S \overset{*}{\underset{G_2}{\Rightarrow}} \alpha X \beta$. Further, every symbol used in this derivation is reachable, so $S \overset{*}{\underset{G_1}{\Rightarrow}} \alpha X \beta$.

We know that every symbol in $\alpha X \beta$ is reachable, and we also know that all these symbols are in $V_2 \cup T_2$, so each of them is generating in G_2. The derivation of some terminal string, say $\alpha X \beta \overset{*}{\underset{G_2}{\Rightarrow}} xwy$, involves only symbols that are reachable from S, because they are reached by symbols in $\alpha X \beta$. Thus, this derivation is also a derivation of G_1; that is,

$$S \overset{*}{\underset{G_1}{\Rightarrow}} \alpha X \beta \overset{*}{\underset{G_1}{\Rightarrow}} xwy$$

We conclude that X is useful in G_1. Since X is an arbitrary symbol of G_1, we conclude that G_1 has no useless symbols.

The last detail is that we must show $L(G_1) = L(G)$. As usual, to show two sets the same, we show each is contained in the other.

$L(G_1) \subseteq L(G)$: Since we have only eliminated symbols and productions from G to get G_1, it follows that $L(G_1) \subseteq L(G)$.

Scanned with CamScanner

$L(G) \subseteq L(G_1)$: We must prove that if w is in $L(G)$, then w is in $L(G_1)$. If w is in $L(G)$, then $S \overset{*}{\underset{G}{\Rightarrow}} w$. Each symbol in this derivation is evidently both reachable and generating, so it is also a derivation of G_1. That is, $S \overset{*}{\underset{G_1}{\Rightarrow}} w$, and thus w is in $L(G_1)$. □

Scanned with CamScanner

Chapter 6

Simplification of Context-Free Grammars and Normal Forms

Before we can study context-free languages in greater depth, we must attend to some technical matters. The definition of a context-free grammar imposes no restriction whatsoever on the right side of a production. However, complete freedom is not necessary and, in fact, is a detriment in some arguments. In Theorem 5.2, we see the convenience of certain restrictions on grammatical forms; eliminating rules of the form $A \to \lambda$ and $A \to B$ make the arguments easier. In many instances, it is desirable to place even more stringent restrictions on the grammar. Because of this, we need to look at methods for transforming an arbitrary context-free grammar into an equivalent one that satisfies certain restrictions on its form. In this chapter we study several transformations and substitutions that will be useful in subsequent discussions.

We also investigate **normal forms** for context-free grammars. A normal form is one that, although restricted, is broad enough so that any grammar has an equivalent normal-form version. We introduce two of the most useful of these, the **Chomsky normal form** and the **Greibach normal form**. Both have many practical and theoretical uses. An immediate application of the Chomsky normal form to parsing is given in Section 6.3.

Scanned with CamScanner

The somewhat tedious nature of the material in this chapter lies in the fact that many of the arguments are manipulative and give little intuitive insight. For our purposes, this technical aspect is relatively unimportant and can be read casually. The various conclusions are significant; they will be used many times in later discussions.

6.1 Methods for Transforming Grammars

We first raise an issue that is somewhat of a nuisance with grammars and languages in general: the presence of the empty string. The empty string plays a rather singular role in many theorems and proofs, and it is often necessary to give it special attention. We prefer to remove it from consideration altogether, looking only at languages that do not contain λ. In doing so, we do not lose generality, as we see from the following considerations. Let L be any context-free language, and let $G = (V, T, S, P)$ be a context-free grammar for $L - \{\lambda\}$. Then the grammar we obtain by adding to V the new variable S_0, making S_0 the start variable, and adding to P the productions

$$S_0 \to S|\lambda$$

generates L. Therefore, any nontrivial conclusion we can make for $L - \{\lambda\}$ will almost certainly transfer to L. Also, given any context-free grammar G, there is a method for obtaining $\hat{G}$ such that $L\left(\hat{G}\right) = L(G) - \{\lambda\}$ (see Exercises 13 and 14 at the end of this section). Consequently, for all practical purposes, there is no difference between context-free languages that include λ and those that do not. For the rest of this chapter, unless otherwise stated, we will restrict our discussion to λ-free languages.

A Useful Substitution Rule

Many rules govern generating equivalent grammars by means of substitutions. Here we give one that is very useful for simplifying grammars in various ways. We will not define the term *simplification* precisely, but we will use it nevertheless. What we mean by it is the removal of certain types of undesirable productions; the process does not necessarily result in an actual reduction of the number of rules.

Theorem 6.1

Let $G = (V, T, S, P)$ be a context-free grammar. Suppose that P contains a production of the form

$$A \to x_1 B x_2.$$

Scanned with CamScanner

Assume that A and B are different variables and that

$$B \to y_1 \,|y_2| \cdots |y_n$$

is the set of all productions in P which have B as the left side. Let $\widehat{G} = \left(V, T, S, \widehat{P}\right)$ be the grammar in which $\widehat{P}$ is constructed by deleting

$$A \to x_1 B x_2 \tag{6.1}$$

from P, and adding to it

$$A \to x_1 y_1 x_2 \,|x_1 y_2 x_2| \cdots |x_1 y_n x_2.$$

Then

$$L\left(\widehat{G}\right) = L(G).$$

Proof: Suppose that $w \in L(G)$, so that

$$S \overset{*}{\Rightarrow}_G w.$$

The subscript on the derivation sign $\Rightarrow$ is used here to distinguish between derivations with different grammars. If this derivation does not involve the production (6.1), then obviously

$$S \overset{*}{\Rightarrow}_{\widehat{G}} w.$$

If it does, then look at the derivation the first time (6.1) is used. The B so introduced eventually has to be replaced; we lose nothing by assuming that this is done immediately (see Exercise 18 at the end of this section). Thus

$$S \overset{*}{\Rightarrow}_G u_1 A u_2 \Rightarrow_G u_1 x_1 B x_2 u_2 \Rightarrow_G u_1 x_1 y_j x_2 u_2.$$

But with grammar $\widehat{G}$ we can get

$$S \overset{*}{\Rightarrow}_{\widehat{G}} u_1 A u_2 \Rightarrow_{\widehat{G}} u_1 x_1 y_j x_2 u_2.$$

Thus we can reach the same sentential form with G and $\widehat{G}$. If (6.1) is used again later, we can repeat the argument. It follows then, by induction on the number of times the production is applied, that

$$S \overset{*}{\Rightarrow}_{\widehat{G}} w.$$

Therefore, if $w \in L(G)$, then $w \in L\left(\widehat{G}\right)$.

Scanned with CamScanner

By similar reasoning, we can show that if $w \in L\left(\hat{G}\right)$, then $w \in L(G)$, completing the proof. ■

Theorem 6.1 is a simple and quite intuitive substitution rule: A production $A \rightarrow x_1 B x_2$ can be eliminated from a grammar if we put in its place the set of productions in which B is replaced by all strings it derives in one step. In this result, it is necessary that A and B be different variables. The case when $A = B$ is partially addressed in Exercises 23 and 24 at the end of this section.

Example 6.1

Consider $G = (\{A, B\}, \{a, b, c\}, A, P)$ with productions

$$A \rightarrow a|aaA|abBc,$$
$$B \rightarrow abbA|b.$$

Using the suggested substitution for the variable B, we get the grammar $\hat{G}$ with productions

$$A \rightarrow a|aaA|ababbAc|abbc,$$
$$B \rightarrow abbA|b.$$

The new grammar $\hat{G}$ is equivalent to G. The string $aaabbc$ has the derivation

$$A \Rightarrow aaA \Rightarrow aaabBc \Rightarrow aaabbc$$

in G, and the corresponding derivation

$$A \Rightarrow aaA \Rightarrow aaabbc$$

in $\hat{G}$.

Notice that, in this case, the variable B and its associated productions are still in the grammar even though they can no longer play a part in any derivation. We will next show how such unnecessary productions can be removed from a grammar.

Removing Useless Productions

One invariably wants to remove productions from a grammar that can never take part in any derivation. For example, in the grammar whose entire production set is

$$S \rightarrow aSb|\lambda|A,$$
$$A \rightarrow aA,$$

the production $S \rightarrow A$ clearly plays no role, as A cannot be transformed into a terminal string. While A can occur in a string derived from S, this

Scanned with CamScanner

can never lead to a sentence. Removing this production leaves the language unaffected and is a simplification by any definition.

Definition 6.1

Let $G = (V, T, S, P)$ be a context-free grammar. A variable $A \in V$ is said to be **useful** if and only if there is at least one $w \in L(G)$ such that

$$S \overset{*}{\Rightarrow} xAy \overset{*}{\Rightarrow} w, \tag{6.2}$$

with x, y in $(V \cup T)^*$. In words, a variable is useful if and only if it occurs in at least one derivation. A variable that is not useful is called **useless**. A production is useless if it involves any useless variable.

Example 6.2

A variable may be useless because there is no way of getting a terminal string from it. The case just mentioned is of this kind. Another reason a variable may be useless is shown in the next grammar. In a grammar with start symbol S and productions

$$\begin{aligned} S &\to A, \\ A &\to aA|\lambda, \\ B &\to bA, \end{aligned}$$

the variable B is useless and so is the production $B \to bA$. Although B can derive a terminal string, there is no way we can achieve $S \overset{*}{\Rightarrow} xBy$.

This example illustrates the two reasons why a variable is useless: either because it cannot be reached from the start symbol or because it cannot derive a terminal string. A procedure for removing useless variables and productions is based on recognizing these two situations. Before we present the general case and the corresponding theorem, let us look at another example.

Example 6.3

Eliminate useless symbols and productions from $G = (V, T, S, P)$, where $V = \{S, A, B, C\}$ and $T = \{a, b\}$, with P consisting of

$$\begin{aligned} S &\to aS|A|C, \\ A &\to a, \\ B &\to aa, \\ C &\to aCb. \end{aligned}$$

Scanned with CamScanner

First, we identify the set of variables that can lead to a terminal string. Because $A \to a$ and $B \to aa$, the variables A and B belong to this set. So does S, because $S \to A \to a$. However, this argument cannot be made for C, thus identifying it as useless. Removing C and its corresponding productions, we are led to the grammar G_1 with variables $V_1 = \{S, A, B\}$, terminals $T = \{a\}$, and productions

$$S \to aS|A,$$
$$A \to a,$$
$$B \to aa.$$

Next we want to eliminate the variables that cannot be reached from the start variable. For this, we can draw a **dependency graph** for the variables. Dependency graphs are a way of visualizing complex relationships and are found in many applications. For context-free grammars, a dependency graph has its vertices labeled with variables, with an edge between vertices C and D if and only if there is a production form

$$C \to xDy.$$

A dependency graph for V_1 is shown in Figure 6.1. A variable is useful only if there is a path from the vertex labeled S to the vertex labeled with that variable. In our case, Figure 6.1 shows that B is useless. Removing it and the affected productions and terminals, we are led to the final answer $\hat{G} = \left(\hat{V}, \hat{T}, S, \hat{P}\right)$ with $\hat{V} = \{S, A\}, \hat{T} = \{a\}$, and productions

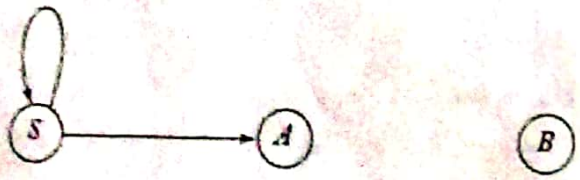
$$S \to aS|A,$$
$$A \to a.$$

The formalization of this process leads to a general construction and the corresponding theorem.

Theorem 6.2

Let $G = (V, T, S, P)$ be a context-free grammar. Then there exists an equivalent grammar $\hat{G} = \left(\hat{V}, \hat{T}, S, \hat{P}\right)$ that does not contain any useless variables or productions.

Figure 6.1



Scanned with CamScanner

First, we identify the set of variables that can lead to a terminal string. Because $A \to a$ and $B \to aa$, the variables A and B belong to this set. So does S, because $S \Rightarrow A \Rightarrow a$. However, this argument cannot be made for C, thus identifying it as useless. Removing C and its corresponding productions, we are led to the grammar G_1 with variables $V_1 = \{S, A, B\}$, terminals $T = \{a\}$, and productions

$$S \to aS|A,$$
$$A \to a,$$
$$B \to aa.$$

Next we want to eliminate the variables that cannot be reached from the start variable. For this, we can draw a **dependency graph** for the variables. Dependency graphs are a way of visualizing complex relationships and are found in many applications. For context-free grammars, a dependency graph has its vertices labeled with variables, with an edge between vertices C and D if and only if there is a production form

$$C \to xDy.$$

A dependency graph for V_1 is shown in Figure 6.1. A variable is useful only if there is a path from the vertex labeled S to the vertex labeled with that variable. In our case, Figure 6.1 shows that B is useless. Removing it and the affected productions and terminals, we are led to the final answer $\hat{G} = \left(\hat{V}, \hat{T}, S, \hat{P}\right)$ with $\hat{V} = \{S, A\}$, $\hat{T} = \{a\}$, and productions

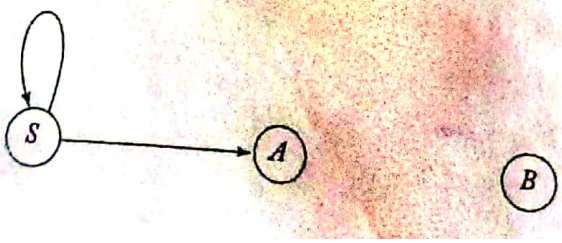
$$S \to aS|A,$$
$$A \to a.$$

The formalization of this process leads to a general construction and the corresponding theorem.

Theorem 6.2

Let $G = (V, T, S, P)$ be a context-free grammar. Then there exists an equivalent grammar $\hat{G} = \left(\hat{V}, \hat{T}, S, \hat{P}\right)$ that does not contain any useless variables or productions.

Figure 6.1



Scanned with CamScanner

Proof: The grammar $\hat{G}$ can be generated from G by an algorithm consisting of two parts. In the first part we construct an intermediate grammar $G_1 = (V_1, T_2, S, P_1)$ such that V_1 contains only variables A for which

$$A \stackrel{*}{\Rightarrow} w \in T^*$$

is possible. The steps in the algorithm are

1. Set V_1 to $\varnothing$.
2. Repeat the following step until no more variables are added to V_1. For every $A \in V$ for which P has a production of the form

$$A \rightarrow x_1 x_2 \cdots x_n, \text{ with all } x_i \text{ in } V_1 \cup T,$$

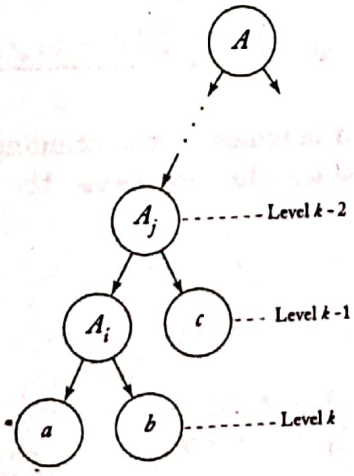
add A to V_1.

3. Take P_1 as all the productions in P whose symbols are all in $(V_1 \cup T)$.

Clearly this procedure terminates. It is equally clear that if $A \in V_1$, then $A \stackrel{*}{\Rightarrow} w \in T^*$ is a possible derivation with G_1. The remaining issue is whether every A for which $A \stackrel{*}{\Rightarrow} w = ab\cdots$ is added to V_1 before the procedure terminates. To see this, consider any such A and look at the partial derivation tree corresponding to that derivation (Figure 6.2). At level k, there are only terminals, so every variable A_i at level $k-1$ will be added to V_1 on the first pass through Step 2 of the algorithm. Any variable at level $k-2$ will then be added to V_1 on the second pass through Step 2. The third time through Step 2, all variables at level $k-3$ will be added, and so on. The algorithm cannot terminate while there are variables in the tree that are not yet in V_1. Hence A will eventually be added to V_1.

In the second part of the construction, we get the final answer $\hat{G}$ from G_1. We draw the variable dependency graph for G_1 and from it find all variables that cannot be reached from S. These are removed from the

Figure 6.2



Scanned with CamScanner

variable set, as are the productions involving them. We can also eliminate any terminal that does not occur in some useful production. The result is the grammar $\hat{G} = \left(\hat{V}, \hat{T}, S, \hat{P}\right)$.

Because of the construction, $\hat{G}$ does not contain any useless symbols or productions. Also, for each $w \in L(G)$ we have a derivation

$$S \overset{*}{\Rightarrow} xAy \overset{*}{\Rightarrow} w.$$

Since the construction of $\hat{G}$ retains A and all associated productions, we have everything needed to make the derivation

$$S \overset{*}{\Rightarrow}_{\hat{G}} xAy \overset{*}{\Rightarrow}_{\hat{G}} w.$$

The grammar $\hat{G}$ is constructed from G by the removal of productions, so that $\hat{P} \subseteq P$. Consequently $L\left(\hat{G}\right) \subseteq L(G)$. Putting the two results together, we see that G and $\hat{G}$ are equivalent. ■

Removing λ-Productions

One kind of production that is sometimes undesirable is one in which the right side is the empty string.

Definition 6.2

Any production of a context-free grammar of the form

$$A \to \lambda$$

is called a **λ-production.** Any variable A for which the derivation

$$A \overset{*}{\Rightarrow} \lambda \tag{6.3}$$

is possible is called **nullable.**

A grammar may generate a language not containing λ, yet have some λ-productions or nullable variables. In such cases, the λ-productions can be removed.

Example 6.4

Consider the grammar

$$S \to aS_1b,$$
$$S_1 \to aS_1b|\lambda,$$

Scanned with CamScanner

with start variable S. This grammar generates the λ-free language $\{a^n b^n : n \geq 1\}$. The λ-production $S_1 \rightarrow \lambda$ can be removed after adding new productions obtained by substituting λ for S_1 where it occurs on the right. Doing this we get the grammar

$$S \rightarrow aS_1b|ab,$$
$$S_1 \rightarrow aS_1b|ab.$$

We can easily show that this new grammar generates the same language as the original one.

In more general situations, substitutions for λ-productions can be made in a similar, although more complicated, manner.

Theorem 6.3

Let G be any context-free grammar with λ not in $L(G)$. Then there exists an equivalent grammar $\widehat{G}$ having no λ-productions.

Proof: We first find the set V_N of all nullable variables of G, using the following steps.

1. For all productions $A \rightarrow \lambda$, put A into V_N.
2. Repeat the following step until no further variables are added to V_N.
   For all productions

$$B \rightarrow A_1A_2 \cdots A_n,$$

   where $A_1, A_2, ..., A_n$ are in V_N, put B into V_N.

Once the set V_N has been found, we are ready to construct $\widehat{P}$. To do so, we look at all productions in P of the form

$$A \rightarrow x_1x_2 \cdots x_m, m \geq 1,$$

where each $x_i \in V \cup T$. For each such production of P, we put into $\widehat{P}$ that production as well as all those generated by replacing nullable variables with λ in all possible combinations. For example, if x_i and x_j are both nullable, there will be one production in $\widehat{P}$ with x_i replaced with λ, one in which x_j is replaced with λ, and one in which both x_i and x_j are replaced with λ. There is one exception: If all x_i are nullable, the production $A \rightarrow \lambda$ is not put into $\widehat{P}$.

The argument that this grammar $\widehat{G}$ is equivalent to G is straightforward and will be left to the reader. ■

Scanned with CamScanner

Example 6.5

Find a context-free grammar without λ-productions equivalent to the grammar defined by

$$S \to ABaC,$$
$$A \to BC,$$
$$B \to b|\lambda,$$
$$C \to D|\lambda,$$
$$D \to d.$$

From the first step of the construction in Theorem 6.3, we find that the nullable variables are A, B, C. Then, following the second step of the construction, we get

$$S \to ABaC\,|BaC|\,AaC\,|ABa|\,aC\,|Aa|\,Ba|a,$$
$$A \to B\,|C|\,BC,$$
$$B \to b,$$
$$C \to D,$$
$$D \to d.$$

Removing Unit-Productions

As we have seen in Theorem 5.2, productions in which both sides are a single variable are at times undesirable.

Definition 6.3

Any production of a context-free grammar of the form

$$A \to B,$$

where $A, B \in V$, is called a **unit-production.**

To remove unit-productions, we use the substitution rule discussed in Theorem 6.1. As the construction in the next theorem shows, this can be done if we proceed with some care.

Theorem 6.4

Let $G = (V, T, S, P)$ be any context-free grammar without λ-productions. Then there exists a context-free grammar $\hat{G} = \left(\hat{V}, \hat{T}, S, \hat{P}\right)$ that does not have any unit-productions and that is equivalent to G.

Scanned with CamScanner

Proof: Obviously, any unit-production of the form $A \to A$ can be removed from the grammar without effect, and we need only consider $A \to B$, where A and B are different variables. At first sight, it may seem that we can use Theorem 6.1 directly with $x_1 = x_2 = \lambda$ to replace

$$A \to B$$

with

$$A \to y_1 \,|y_2| \cdots |y_n.$$

But this will not always work; in the special case

$$A \to B,$$
$$B \to A,$$

the unit-productions are not removed. To get around this, we first find, for each A, all variables B such that

$$A \overset{*}{\Rightarrow} B. \tag{6.4}$$

We can do this by drawing a dependency graph with an edge (C, D) whenever the grammar has a unit-production $C \to D$; then (6.4) holds whenever there is a walk between A and B. The new grammar $\widehat{G}$ is generated by first putting into $\widehat{P}$ all non-unit productions of P. Next, for all A and B satisfying (6.4), we add to $\widehat{P}$

$$A \to y_1 \,|y_2| \cdots |y_n,$$

where $B \to y_1 \,|y_2| \cdots y_n$ is the set of all rules in $\widehat{P}$ with B on the left. Note that since $B \to y_1 \,|y_2| \cdots |y_n$ is taken from $\widehat{P}$, none of the y_i can be a single variable, so that no unit-productions are created by the last step.

To show that the resulting grammar is equivalent to the original one, we can follow the same line of reasoning as in Theorem 6.1. ■

Example 6.6

Remove all unit-productions from

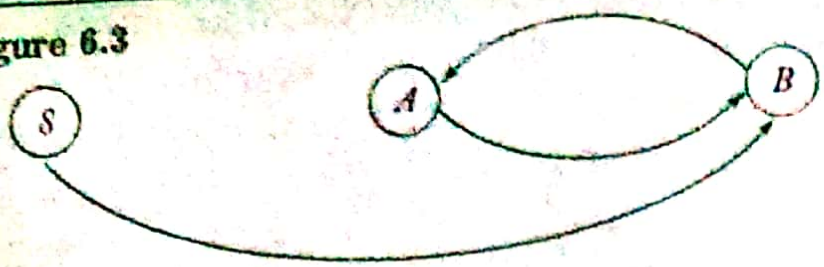
$$S \to Aa|B,$$
$$B \to A|bb,$$
$$A \to a\,|bc|\,B.$$

The dependency graph for the unit-productions is given in Figure 6.3; we see from it that $S \overset{*}{\Rightarrow} A, S \overset{*}{\Rightarrow} B, B \overset{*}{\Rightarrow} A$, and $A \overset{*}{\Rightarrow} B$. Hence, we add to the original non-unit productions

$$S \to Aa,$$
$$A \to a|bc,$$
$$B \to bb,$$

Scanned with CamScanner

Figure 6.3



the new rules

$$S \to a\,|bc|\,bb,$$
$$A \to bb,$$
$$B \to a|bc,$$

to obtain the equivalent grammar

$$S \to a\,|bc|\,bb|Aa,$$
$$A \to a\,|bb|\,bc,$$
$$B \to a\,|bb|\,bc.$$

Note that the removal of the unit-productions has made B and the associated productions useless.

We can put all these results together to show that grammars for context-free languages can be made free of useless productions, λ-productions, and unit-productions.

Scanned with CamScanner

6.2 Two Important Normal Forms

There are many kinds of normal forms we can establish for context-fre grammars. Some of these, because of their wide usefulness, have been stud ied extensively. We consider two of them briefly.

Chomsky Normal Form

One kind of normal form we can look for is one in which the number of symbols on the right of a production is strictly limited. In particular, we can ask that the string on the right of a production consist of no more than two symbols. One instance of this is the **Chomsky normal form.**

Definition 6.4

A context-free grammar is in Chomsky normal form if all productions are of the form

$$A \to BC$$

or

$$A \to a,$$

where A, B, C are in V, and a is in T.

Scanned with CamScanner

Example 6.7

The grammar

$$S \to AS|a,$$
$$A \to SA|b$$

is in Chomsky normal form. The grammar

$$S \to AS|AAS,$$
$$A \to SA|aa$$

is not; both productions $S \to AAS$ and $A \to aa$ violate the conditions of Definition 6.4.

Theorem 6.6

Any context-free grammar $G = (V, T, S, P)$ with $\lambda \notin L(G)$ has an equivalent grammar $\hat{G} = \left(\hat{V}, \hat{T}, S, \hat{P}\right)$ in Chomsky normal form.

Proof: Because of Theorem 6.5, we can assume without loss of generality that G has no λ-productions and no unit-productions. The construction of $\hat{G}$ will be done in two steps.

Step 1: Construct a grammar $G_1 = (V_1, T, S, P_1)$ from G by considering all productions in P in the form

$$A \to x_1 x_2 \cdots x_n, \tag{6.5}$$

where each x_i is a symbol either in V or in T. If $n = 1$, then x_1 must be a terminal since we have no unit-productions. In this case, put the production into P_1. If $n \geq 2$, introduce new variables B_a for each $a \in T$. For each production of P in the form (6.5) we put into P_1 the production

$$A \to C_1 C_2 \cdots C_n,$$

where

$$C_i = x_i \text{ if } x_i \text{ is in } V,$$

and

$$C_i = B_a \text{ if } x_i = a.$$

For every B_a we also put into P_1 the production

$$B_a \to a.$$

This part of the algorithm removes all terminals from productions whose right side has length greater than one, replacing them with newly introduced

Scanned with CamScanner

variables. At the end of this step we have a grammar G_1 all of whose productions have the form

$$A \to a, \tag{6.6}$$

or

$$A \to C_1 C_2 \cdots C_n, \tag{6.7}$$

where $C_i \in V_1$.

It is an easy consequence of Theorem 6.1 that

$$L(G_1) = L(G).$$

Step 2: In the second step, we introduce additional variables to reduce the length of the right sides of the productions where necessary. First we put all productions of the form (6.6) as well as all the productions of the form (6.7) with $n = 2$ into $\hat{P}$. For $n > 2$, we introduce new variables $D_1, D_2, \ldots$ and put into $\hat{P}$ the productions

$$\begin{aligned} A &\to C_1 D_1, \\ D_1 &\to C_2 D_2, \\ &\vdots \\ D_{n-2} &\to C_{n-1} C_n. \end{aligned}$$

Obviously, the resulting grammar $\hat{G}$ is in Chomsky normal form. Repeated applications of Theorem 6.1 will show that $L(G_1) = L\left(\hat{G}\right)$, so that

$$L\left(\hat{G}\right) = L(G).$$

This somewhat informal argument can easily be made more precise. We will leave this to the reader. ■

Example 6.8

Convert the grammar with productions

$$\begin{aligned} S &\to ABa, \\ A &\to aab, \\ B &\to Ac \end{aligned}$$

to Chomsky normal form.

Scanned with CamScanner

As required by the construction of Theorem 6.6, the grammar does not have any λ-productions or any unit-productions.

In Step 1, we introduce new variables B_a, B_b, B_c and use the algorithm to get

$$\begin{aligned} S &\to ABB_a, \\ A &\to B_aB_aB_b, \\ B &\to AB_c, \\ B_a &\to a, \\ B_b &\to b, \\ B_c &\to c. \end{aligned}$$

In the second step, we introduce additional variables to get the first two productions into normal form and we get the final result

$$\begin{aligned} S &\to AD_1, \\ D_1 &\to BB_a, \\ A &\to B_aD_2, \\ D_2 &\to B_aB_b, \\ B &\to AB_c, \\ B_a &\to a, \\ B_b &\to b, \\ B_c &\to c. \end{aligned}$$

Scanned with CamScanner