7.1 Context-free Grammar

Now, let us see **"What is Context-free grammar?"** The Context-free Grammar also called type 2 grammar is defined as follows.

Definition: A context-free grammar (CFG) G is a 4-tuple (or quadruple) denoted by

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

where

- V is set of variables
- T is set of terminals
- P is set of productions (also called rules)
- S is the start symbol

Ex 1: Grammar to generate one or more a's is shown below:

$A \rightarrow a \mid aA$

Ex 2: Grammar to recognize an if-statement is shown below:

$S \rightarrow i\ C\ t\ S \mid i\ C\ t\ S\ e\ S \mid a$

$C \rightarrow b$

where *i* – stands for **if** keyword

t – stands for **the n** keyword

e – stands for **else** keyword

a – stands for a statement

b – stands for a statement

In every context-free grammar, observe the following points:

- In a CFG (Context-free Grammar) all productions are of the form $A \rightarrow \alpha$ where $\alpha \in (V \cup T)^*$ and A is non-terminal.
- The symbol ε can appear on the right-hand side of any production.
- 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.

7.2 Derivation

Now, let us see **"What is derivation?"**

Definition: The process of obtaining string of terminals and/or non-terminals from the start symbol by applying some set of productions (it may include all productions) is called ***derivation.*** Observe the following rules followed during derivation:

- The derivation starts with start symbol S and ends in string of terminals.

- At each step in the derivation process only one variable is replaced by selecting appropriate production among various productions defined in grammar.

For example, consider the following two productions

$$A \rightarrow \alpha B\gamma$$

$$B \rightarrow \beta$$

where

- α, β and γ and are strings of terminals and/or non-terminals
- A and B are non-terminals.

The non-terminal A produces the string "$\alpha\beta\gamma$" as shown below:

$A \Rightarrow \alpha$ // A is replaced by $\alpha B\gamma$ using $A \rightarrow \alpha B\gamma$

$\Rightarrow \alpha\beta\gamma$ // B is replaced by β using $B \rightarrow \beta$

The above two derivations can be written in a single line as shown below:

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

Observe the following points:

- 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: There is a difference between the arrow used in the production and the double arrow used in the derivation:

- We use the single arrow system '$\rightarrow$' to define a grammar rule in the production
- We use double arrow symbol "$\Rightarrow$" to construct the string in the derivation.

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

Example 7.1:

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**

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.

$E \Rightarrow E + E$ // using the production $E \rightarrow E + E$

$\Rightarrow$ **id** + E // using the production $E \rightarrow$ **id**

$\Rightarrow$ **id** + E * E // using the production $E \rightarrow E * E$

$\Rightarrow$ **id** + **id** * E // using the production $E \rightarrow$ **id**

$\Rightarrow$ **id** + **id** * **id** // using the production $E \rightarrow$ **id**

Thus, the above sequence of steps can also be written as:

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

which indicates that the string **id + id * id** is obtained in one or more steps by applying vario productions.

Now, let us see **"What are the two types of derivations?"** The two types of derivations are

- Leftmost derivation
- Rightmost derivation

7.2.1 Leftmost Derivation

Now, let us see **"What is leftmost derivation?"**

Definiation: The process of obtaining a string of terminals from a sequence of replacements such that only leftmost non-terminal is replaced at each and every step is called **leftmost derivation.**

Example 7.2: Consider the following grammar:

$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow (E)$

$E \rightarrow$ **id**

Obtain the leftmost derivation and rightmost for the string **id + id * id**

Solution:

The string **id + id * id** can be obtained using leftmost derivation as shown below:

E E + E //using the production $E \rightarrow E + E$

$\Rightarrow$ **id** + E // Replacing leftmost E by **id**

$\Rightarrow$ **id** + E * E // Replacing leftmost E by E*E

$\Rightarrow$ **id** + **id** * E // Replacing leftmost E by **id**

$\Rightarrow$ **id** + **id** * **id** // Replacing leftmost E by **id**

Example 7.3: Obtain the leftmost derivation for the string aaabbabbba using the following grammar.

S $\rightarrow$ aB| bA

A $\rightarrow$ aS | bAA | a

B $\rightarrow$ bS | aBB | b

Solution:

The leftmost derivation for the string aaabbabbba is shown below:

$S \overset{lm}{\Rightarrow} aB$ (Applying $S \rightarrow aB$)

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

$\Rightarrow aaaBBB$ (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 $A \rightarrow a$)

7.2.2 Rightmost Derivation

Now, let us see **"What is rightmost derivation?"**

Definition: The process of obtaining a string of terminals from a sequence of replacements such that only right most non-terminal is replaced at each and every step is called **rightmost derivation**.

For example, consider the following grammar:

$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow (E)$

$E \rightarrow$ **id**

The rightmost derivation for the string **id + id * id** can be obtained as shown below:

$E \overset{rm}{\Rightarrow} E + E$ // using the production $E \rightarrow E + E$

$\Rightarrow E + E * E$ // Replacing rightmost E by E*E

$\Rightarrow E + E *$ **id** // Replacing rightmost E by **id**

$\Rightarrow E +$ **id * id** // Replacing rightmost E by **id**

$\Rightarrow$ **id + id * id** // Replacing rightmost E by **id**

7.3 Sentence

In this section, let us see **"What is a sentence?"**

Definition: Let G = (V, T, P, S) be a Context-free Grammar. A string $w \in (V \cup T)^*$ which is derivable from the start symbol S such that $S \overset{*}{\Rightarrow} w$ is called a **sentence** or ***sentential form*** of G. For example, consider the derivation:

7.4 Grammars for Other Languages

In this section, let us see the method of writing grammars for various languages.

Example 7.5: Obtain a grammar to generate the following language:

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

Solution:

The following grammar generates any number of a's (see the previous chapter).

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

Now, for every *a* we need to generate on *b*. This is obtained by suffixing aS with one *b* as shown below:

$$S \rightarrow aSb$$

So, the grammar to generate the language $L = \{a^n b^n : n \geq 0\}$ is:

$$\boxed{S \rightarrow \varepsilon \mid aSb}$$

Example 7.6: 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* in the previous example. So, the final grammar is given by:

$$\boxed{S \rightarrow ab \mid aSb}$$

Example 7.7: Obtain a grammar to generate the following language:

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

Solution:

It is similar to the example 7.5. But, one extra *a* should be generated. This is achieved by replacing ε with *a*. So, the final grammar to generate the given language is:

$$\boxed{S \rightarrow a \mid aSb}$$

Example 7.8: Obtain a grammar to generate the following language:

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

Solution:

It is similar to the previous example. But, instead of generating one extra *a* we need generate one extra *b*. So, the final grammar to generate the given language is:

$$\boxed{S \rightarrow b \mid aSb}$$

Example 7.9: Obtain a grammar to generate the following language:

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

Solution:

It is similar to the previous example. But, instead of generating one extra, two extra *b*'s should be generated. So, the final grammar to generate the given language is:

$$\mathbf{S \rightarrow bb \mid aSb}$$

Example 7.10: Obtain a grammar to generate the following language:

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

Solution:

Consider the grammar shown in example 7.5:

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

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

$$\mathbf{S \rightarrow \varepsilon \mid aSbb}$$

Example 7.11: Obtain a grammar G generating set of all palindromes over $\Sigma = \{a, b\}$

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

Solution:

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

- ε is a palindrome. The equivalent productions is $\mathbf{S \rightarrow \varepsilon}$
- *a* and *b* are palindromes. The equivalent productions are $\mathbf{S \rightarrow a \mid b}$
- If *w* is a palindrome then the string *awa* and the string *bwb* are palindromes. So, if S is a palindrome, aSa and bSb are palindromes. The equivalent productions are:

  $$S \rightarrow aSa \mid bSb$$

So, the final grammar can be written as shown below:

$S \rightarrow \varepsilon$ [Definition 1]

$S \rightarrow a \mid b$ [Definition 2]

$S \rightarrow aSa \mid bSb$ [Definition 3]

Example 7.12: Obtain a grammar to generate $L = \{ww^R \text{ where } w \in \{a, b\}^*\}$

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 achieved by deleting the productions $S \to a \mid b$ in the previous grammar. So, the final grammar is given by:

$$S \to \varepsilon$$
$$S \to aSa \mid bSb$$

Note: In the above grammar if the production

$$S \to \varepsilon$$

is replaced by

$$S \to c$$

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

Example 7.13: 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 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

$$S \to aSa \mid bSb \mid A$$

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

$$A \to aBb \mid bBa$$

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

$$B \to aB \mid bB \mid \varepsilon$$

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

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

$S \to aSa \mid bSb$		[Generates palindromes both on left side and right side]
$S \to A$		[Generates a non-palindrome]
$A \to aBb \mid bBa$		[Generates a non palindrome

$B \rightarrow aB \mid bB \mid \varepsilon$ [with B generating any number of a's and b's]
[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 7.14: Obtain the grammar to generate $L = \{0^m1^m2^n \mid m \geq 1 \text{ and } n \geq 0\}$

Solution:

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

$$L = \{\underbrace{0^m1^m}\underbrace{2^n} \mid m \geq 1 \text{ and } n \geq 0\}$$
$$S \rightarrow A \quad B \qquad \ldots(1)$$

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 7.4]
- B should produce any number of 2's. Any number of 2's can be generated using the production:
  $B \rightarrow \varepsilon \mid 2B \mid$ [Similar to example 5.1]

So, final grammar to accept the given language is:

$S \rightarrow A \ B$
$A \rightarrow 01 \mid 0A1$
$B \rightarrow \varepsilon \mid 2B$

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 7.15: Obtain the grammar to generate the language

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

Solution: The given language can also be written as shown below:

$$L = \{w \mid \#_a(w) = \#_b(w)\}$$

$n_a(w)$ is same as $\#_a(w)$ and $n_b(w)$ is same as $\#_b(w)$. Here, $n_a(w) = n_b(w)$ 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:

- An empty string denoted by ε has equal number of a's and b's (i.e., zero a's and zero b's). The production can be written as:

$$S \rightarrow \varepsilon$$

- The symbol 'a' can be followed by the symbol 'b'. The equivalent production can be written as:

$$S \rightarrow aSb$$

- The symbol 'b' can be followed by the symbol 'a'. The equivalent production can be written as:

$$S \rightarrow bSa$$

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

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

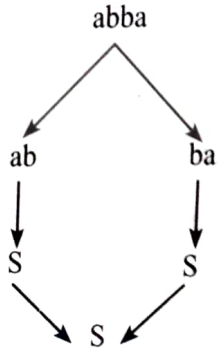
$\Rightarrow ab$ (By applying $S \rightarrow \varepsilon$)

Similarly, the sub string 'ba' can be generated from the 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

$$S \rightarrow SS$$

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

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

Example 7.16: 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 $S \rightarrow \varepsilon$ 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 production $S \rightarrow 0A$ and $A \rightarrow 1S$ and finally applying the produciton $S \rightarrow \varepsilon$, 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:

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

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

$S \rightarrow 01S \mid \varepsilon$
or
$S \rightarrow S01 \mid \varepsilon$

Example 7.17: Obtain a CFG to generate a string of balanced parentheses.

Solution:

It is similar to the example 7.5 which generates the language a^nb^n. The changes to be done for this solution are:

- If we replace the symbol *a* with ' (', then symbol *b* is replaced by ') '. The equivalent production is:

  $S \rightarrow (S)$

- If we replace the symbol *a* with ' (', then symbol *b* is replaced by ') '. The equivalent production is:

  $S \rightarrow [S]$

- If we replace the symbol *a* with ' (', then symbol *b* is replaced by ') '. The equivalent production is:

  $S \rightarrow \{S\}$

- The strings of the form "() ()" or "[] []" can be generated using the production:

  $S \rightarrow SS$

So, the final grammar 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 7.18: Obtain a grammar to generate the $L = \{0^i1^j \mid i \neq j, i \geq 0 \text{ and } j \geq 0\}$

Solution:

We should generate 0's followed by 1's such that number of zeros should not be equal to the number of 1's. If we apply the production:

$S \rightarrow 0S1$

repeatedly we get a string of the form:

0^nS1^n

Now, if we can replace S with A from which we can generate at least one extra 0 as shown below:

$$A \rightarrow 0 \mid 0A$$

we are able to generate the given language $L = \{0^i1^j \mid i \neq j, i \geq 0 \text{ and } j \geq 0\}$

Also, if we can replace S with B from which we can generate at least one extra 1 as shown below:

$$B \rightarrow 1 \mid 1B$$

we are able to generate the given language $L = \{0^i1^j \mid i \neq j, i \geq 0 \text{ and } j \geq 0\}$. So, the final grammar is:

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

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]
S	→	0A | 0	[At least one extra 0 is generated]
S	→	1B | 1	[At least one extra 1 is generated]

$\}$
S is the start symbol

Another solution to generate the same language can be written as

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

where

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

S	→	AC |	[At least one 0 is preceded by 0^n1^n]
S	→	CB	[At least one 1 is followed by $^0n1^n$]
C	→	0C1 | ε	[Generates $0^n1^n \mid n \geq 0$]
A	→	0A | 0	[At least one 0 is generated]
B	→	1B | 1	[At least one 1 is generated]

$\}$
S is the start symbol

Example 7.19: Obtain a grammar to generate the $L = \{a^{n+2}b^m \mid n \geq 0 \text{ and } m > n\}$

Solution:

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

$n = 0 \quad n = 1 \quad n = 2$

$m > 0 \quad m > 1 \quad m > 2$

i.e., $m \geq 1 \quad m \geq 2 \quad m \geq 3$

$L = \{\mathbf{ab}b^*, \mathbf{aabb}b^*, \mathbf{aaabbb}b^*, \ldots\ldots\ldots\}$

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]

Example 7.20: Obtain a grammar to generate $L = \{a^nb^m \mid n \geq 0, m > n\}$

Solution:

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

$n = 0 \quad n = 1 \quad n = 2$

$m \geq 1 \quad m \geq 2 \quad m \geq 3$

$L = \{ \quad \varepsilon bb^*, \quad \mathbf{ab}bb^*, \quad \mathbf{aabb}bb^*, \ldots\ldots\ldots\}$

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

$$S \rightarrow AB$$

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

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

and one or more *b*'s sare generated from the production:

$$A \rightarrow bB \mid b$$

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

V = {S, A, B}
T = {a, b}
P = {
S → AB
A → aAb | ε [Generates $a^nb^n \mid n \geq 0$
B → bB | b [Generates one or more b's]
}
S - is the start symbol

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

S → aSb | B [Generates $a^nb^n \mid n \geq 0$ followed by at least one b]
B → bB | b [Generates one or more b's]

Example 7.21: Obtain a grammar to generate the language $L = \{a^nb^{n-3} \mid n \geq 3\}$

Solution:

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

n = 3 n = 4 n = 5 n = 6

L = {aaa, aaaab, aaaaabb, aaaaaabbb, ...}

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

S → aaaA

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

A → aAb | ε

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

V = {S, A}
T = {a, b}
P = {
S → aaaA
S → aAb | ε
}
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 7.22: 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 7.18. For convenience it is provided again:

$S1 \rightarrow aS_1b \mid B$

$B \rightarrow bB \mid b$

The grammar corresponding to the language L_2 is already obtained in example 7.8. For convenience it is provided again:

$S_2 \rightarrow 0S_2 11 \mid \varepsilon$

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:

$V = \{S, S_1, S_2, B\}$

$T = \{a, b, 0, 1\}$

$P = \{$

$S \rightarrow S_1S_2$

$S_1 \rightarrow aS_1b \mid bB$

$S_2 \rightarrow 0S_2 11 \mid \varepsilon$

$B \rightarrow bB \mid b$

$\}$

S_1 - is the start symbol

Example 7.23: Obtain a grammar to generate the language $L = L_1 \cup L_2$ where

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

Solution:

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.

$S \rightarrow S_1 \mid S_2$

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

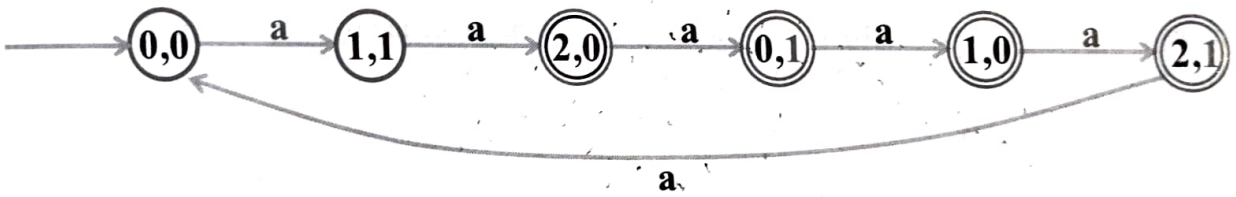
$V = \{S, S_1, S_2, A, B\}$
$T = \{a, b, 0, 1\}$
$P = \{$
$S \rightarrow S_1 \mid S_2$
$S_1 \rightarrow AB$
$S_2 \rightarrow 0S_211 \mid \varepsilon$
$A \rightarrow aAb \mid \varepsilon$
$B \rightarrow bB \mid b$
$\}$
S_1 - is the start symbol

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

Example 7.24: Obtain a grammar to generate $L = \{w : |w| \bmod 3 \neq |w| \bmod 2\}$ on $\Sigma = \{a\}$

Solution:

The DFA to accept the above language can be written as shown below (see example 2.45 for details)



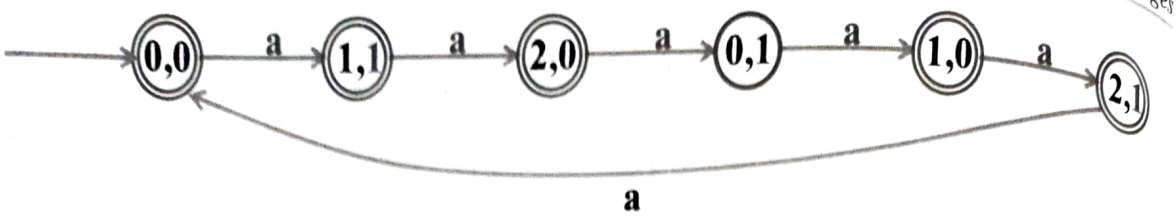
Using the above DFA, the grammar to accept the given language can be written as shown below:

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

Example 7.25: Obtain a grammar to generate $L = \{w : |w| \bmod 3 \geq |w| \bmod 2\}$ on $\Sigma = \{a\}$

Solution:

The DFA to accept the above language can be written as shown below (see example 2.45 for details)



Using the above DFA, the grammar to accept the given language can be written as shown below:

V = {S}
T = {a}
P = {
S → ε | a | aa | aaaa | aaaaa | aaaaaaS
}
S - is the start symbol

Example 7.26: Obtain a grammar to generate set of all strings with exactly one a when $\Sigma = \{a, b\}$

Solution:

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 → bS | 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 → bB | ε

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

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

Example 7.27: Obtain a grammar generating all strings with at least one *a* if $\Sigma = \{a, b\}$

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

$S \rightarrow bS$

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

Solution:

The regular expression representing not more than three *a*'s can be written as shown below:

$$(\varepsilon + a)\ (\varepsilon + a)\ (\varepsilon + a)$$

But, there is no restriction on number of b's and the position where b's occur. Any number of b's can be inserted in the beginning/middle or at the end as shown below:

$$b^*(\varepsilon + a)\ b^*\ (\varepsilon + a)\ b^*\ (\varepsilon + a)\ b^*$$

Now, the grammar can be written as shown below:

$$b^*(\varepsilon + a)\ b^*\ (\varepsilon + a)\ b^*\ (\varepsilon + a)\ b^*$$
$$S \rightarrow B\ A\ B\ A\ B\ A\ B$$

where B generates any number of a's using the production:

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

and a generates either *a* or ε, using the production:

$$A \rightarrow \varepsilon \mid a$$

Now, the final grammar can be written as shown below:

$S \rightarrow BABABAB$
$B \rightarrow \varepsilon \mid bB$
$A \rightarrow \varepsilon \mid a$

Alternate solution: Based on number of a's following four cases can be considered:

- there can be no *a*'s at all
- there can be only one *a*
- there can be only two *a*'s
- there can be only three *a*'s

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

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

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 the non-terminal A, we can generate any number of b's using the production.

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

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

$$A \rightarrow aB$$

From B, we can generate any number of b's using the production:

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

Case 4: There can be three *a*'s. The moment we get B in the derivation, we will have two a's. A third *a* can be generated from B using the following production:

$$B \rightarrow aC$$

From variable C, we can get any number of b's. The productions to generate any number of b's from non-terminal C can be written as shown below:

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

So, the final grammar can be written as shown below:

S	$\rightarrow bS \mid aA \mid \varepsilon$
A	$\rightarrow bA \mid aB \mid \varepsilon$
B	$\rightarrow bB \mid aC \mid \varepsilon$
C	$\rightarrow bC \mid \varepsilon$

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

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

where any number of b's can be generated from the non-terminal B as shown below:

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

So, the final grammar to generate the given language can be written as shown below:

$$S \rightarrow B \mid BaB \mid BaBaB \mid BaBaBaB$$
$$B \rightarrow bB \mid \varepsilon$$

Example 7.29: Obtain a grammar to generate the language $L = \{w \mid n_a(w) = n_b(w) + 1\}$

Solution:

From example 7.15, the grammar to generate equal number of a's and b's can be written as shown below:

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

But, according to the problem, we need to generate one extra *a*. This extra *a* can be present either in the beginning or at the end or at the middle. This can be achieved using the production

$$S \rightarrow AaA$$

So, the final grammar to generate the language: $L = \{w \mid n_a(w) = n_b(w) + 1\}$ can be written as shown below:

$$S \rightarrow AaA$$
$$A \rightarrow aAb \mid bAa \mid AA \mid \varepsilon$$

Example 7.30: 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, we should be in a position to generate 1 or more a's. One or more a's can be generated using the production:

$$B \rightarrow aB \mid a$$

Now, at least one extra *a* generated may be at the end of equal number of a's and b's. The production to achieve this case can be written as shown below:

$$S \rightarrow AB$$

Equal number of a's and b's can be preceded by one or more a's which can be achieved by introducing the following 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 $L = \{w \mid n_a(w) > n_b(w)\}$ can be written as shown below:

S	→	AB|BA|ABA
A	→	aAb
A	→	bAa
A	→	AA
A	→	ε
B	→	aB | a

Example 7.31: Obtain a grammar to generate a language of strings of 0's and 1's having a substring 000

Solution:

The given language can be written as shown below:

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

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 can be written as shown below:

S	→	A000A
A	→	0A | 1A | ε

Example 7.32: Obtain a grammar to generate $L = \{a^n b^m c^k \mid n + 2m = k \text{ for } n \geq 0, m \geq 0\}$

Solution:

It is required to generate the grammar for the language:

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

It is given that $k = n + 2m$. Substituting for k in the language:

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

the above language can be written as shown below:

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

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 below:

$$S \rightarrow aSc \mid A$$

So, the final grammar can be written as shown below:

$$S \rightarrow aSc \mid A$$
$$A \rightarrow bAcc \mid \varepsilon$$

Example 7.34: Obtain a grammar to generate $L = \{a^n b^n c^m : n, m \geq 0\}$

Solution:

The production to generate the above language can be written as shown below:

$$S \rightarrow \underbrace{A}_{a^n b^n} \; \underbrace{C}_{c^m}$$

From non-terminal A we should generate *n* number of a's followed by n number of b's and from B we should generate any number of c's. The productions to generate *n* number of a's followed by *n* number of b's (explained already in example 7.3) can be written as shown below:

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

Any number of c's can be generated using the following production:

$$C \rightarrow \varepsilon \mid cC$$

Now, the final grammar can be generated as shown below:

S	→	A C
A	→	aAb | ε
C	→	ε | cC

Example 7.35: Obtain a grammar to generate $L = \{a^{n1} b^{n1} a^{n2} b^{n2} \ldots\ldots a^{nk} b^{nk} : n, k \geq 0\}$

Solution:

The productions to generate *n* number of a's followed by *n* number of b's can be generated using the productions:

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

and k number of A's can be generated using the production:

$$S \rightarrow AS \mid \varepsilon$$

So, the final grammar can be generated using the following production:

$$\boxed{\begin{array}{lcl} S & \rightarrow & AS \mid \varepsilon \\ A & \rightarrow & aAb \mid \varepsilon \end{array}}$$

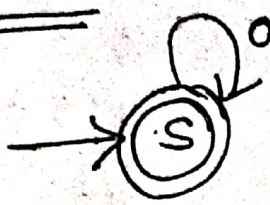
Grammar from Finite Automata

(1) Obtain grammar to generate string consisting of any number of a's. or $L=\{a^n : n \geq 0\}$

Sol²:- Any no. of a's

$L=\{\epsilon, a, aa, aaa, aaaa, \ldots\}$

DFA



S is a final state ---> $S \to \epsilon$

$\delta(S,a)=S$ ----> $S \to aS$

P: $S \to aS$, $S \to \epsilon$ Production rules,

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

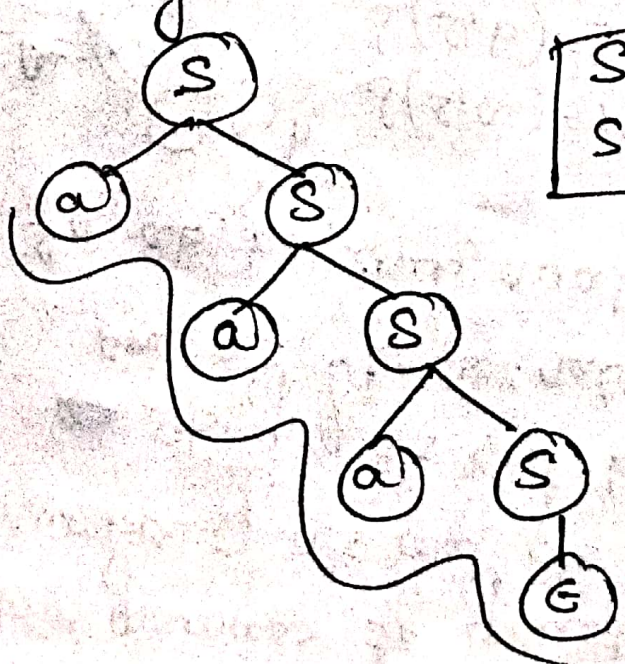
$V=\{S\}$ → non terminals

$T=\{a\}$ → Terminals

$S=S$ → start symbol.

Verification - Parse tree.

String = aaa, use production rules



$S \to aS$, $S \to \epsilon$

↓

verified.

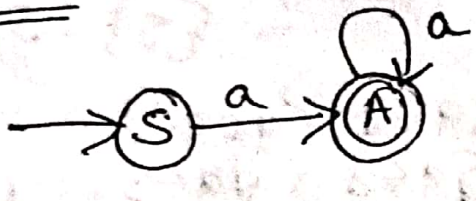
2] obtain grammar to generate string consisting of at least one a. or $L=\{a^n : n \geq 1\}$

Sol:

Atleast one a

$L=\{a, aa, aaa, aaaa, ---\}$

DFA:



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

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

A is final state $\rightarrow A \rightarrow \epsilon$

(V, T, P, S)

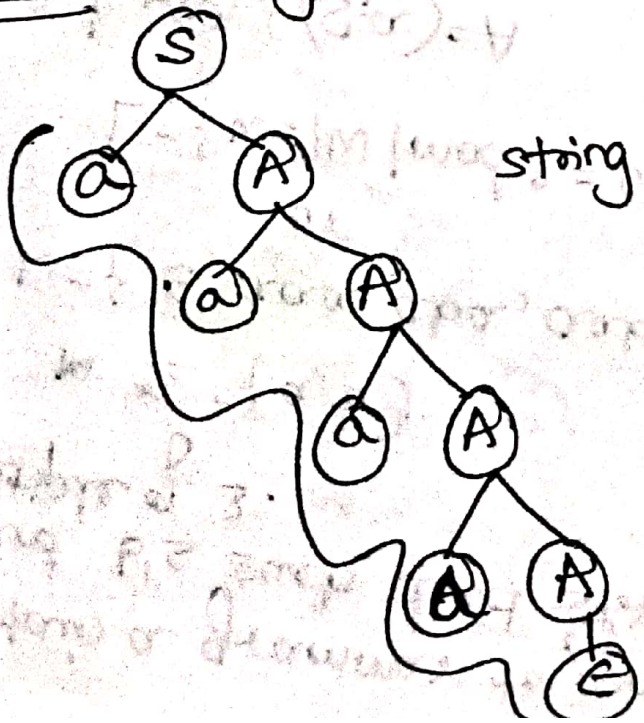
$V=\{S, A\}$

$T=\{a\}$

$P=\{S \rightarrow aA,$
$A \rightarrow aA,$
$A \rightarrow \epsilon\}$

$S=\{S\}$ → start symbol.

verification - string = aaaa

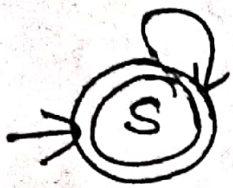


string is verified.

(3) Obtain a grammar to generate string consisting of any number of a's and b's.

Sol^n - any no. of a's and b's

$L = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$



$\delta(S,a) = S \longrightarrow S \to aS$

$\delta(S,b) = S \longrightarrow S \to bS$

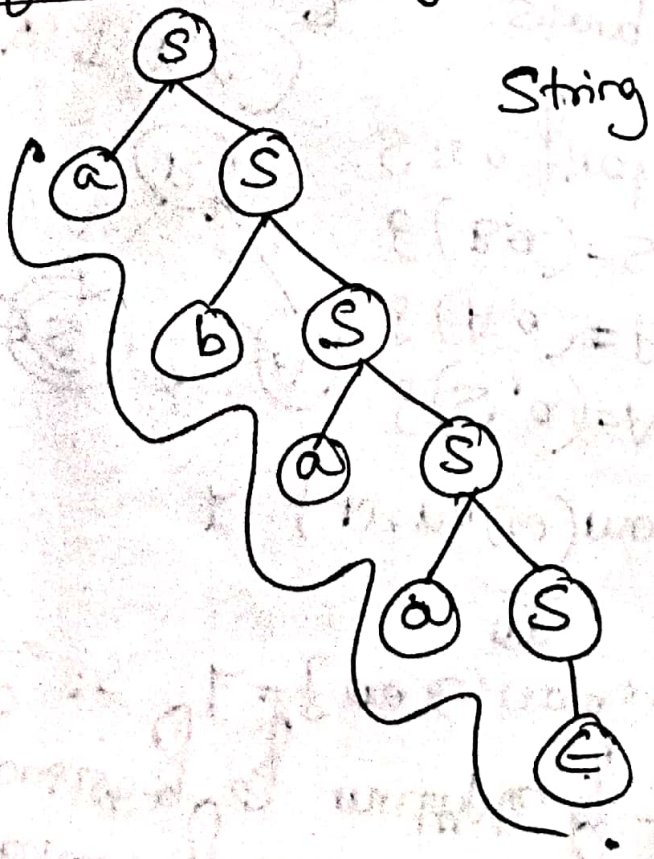
S is a final state $\longrightarrow S \to \epsilon$

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

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

$P = \{ S \to aS,$
$S \to bS,$
$S \to \epsilon \}$

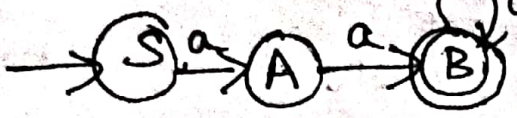
Verification :- String = abaa



String is verified.

4) obtain a grammar to generate string consisting of at least two a's.

Sol^n at least two a's



$L = \{aa, aaa, aaaa, \ldots\}$

$\delta(S, a) = A \longrightarrow S \to aA$

$\delta(A, a) = B \longrightarrow A \to aB$

$\delta(B, a) = B \longrightarrow B \to aB$

B is a final state $\to B \to \epsilon$

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

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

$T = \{a\}$

$P = \{ S \to aA,$
$A \to aB,$
$B \to aB,$
$B \to \epsilon \}$

$S = \{S\}$

or

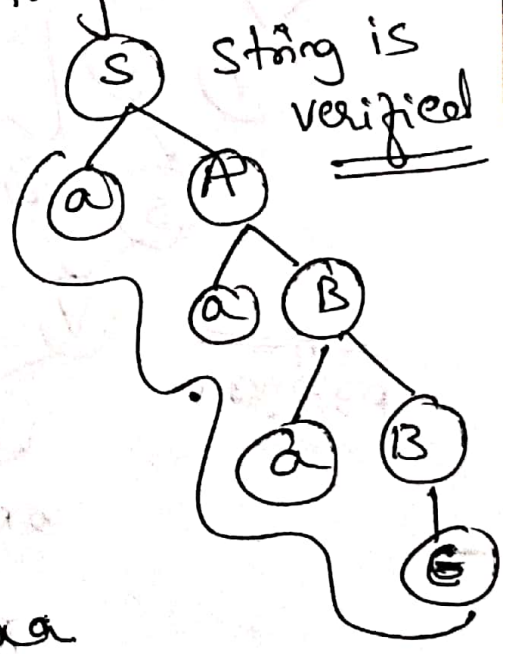
Another Sol^n –

$S \to aa$

$S \to aS.$

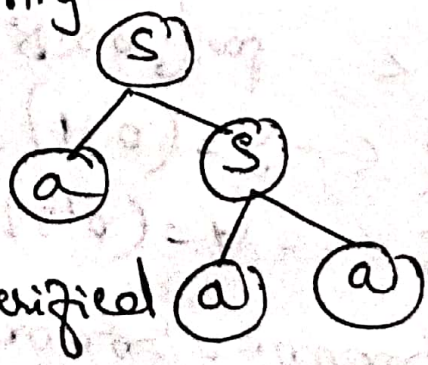
Verification

String = aaa



String is verified

string = aaa

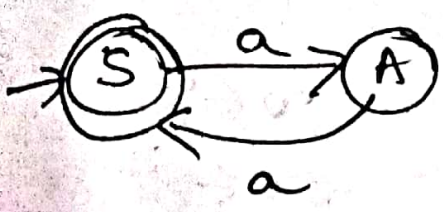


String is verified.

5] obtain grammar to generate string consisting of even no. of a's or $L=\{n_a(w) \bmod 2 = 0\}$

sol^n. even no. of a's

$L=\{\epsilon, aa, aaaa, aaaaaa, \dots\}$



$\delta(S,a) = A \longrightarrow S \to aA$

$\delta(A,a) = S \longrightarrow A \to aS$

S is a final state $\longrightarrow S \to \epsilon$

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

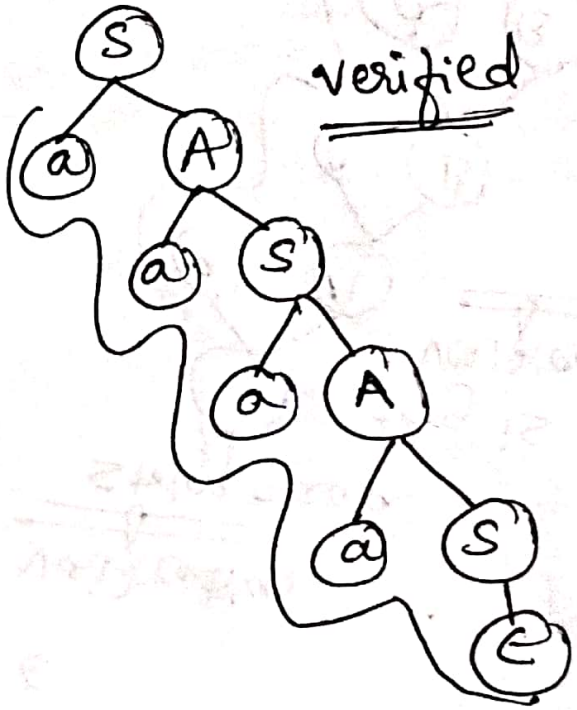
$V = \{S, A\}$

$T = \{a\}$

$P = \{ S \to aA,$
$A \to aS,$
$S \to \epsilon \}$

$S = \{S\}$

string: aaaa



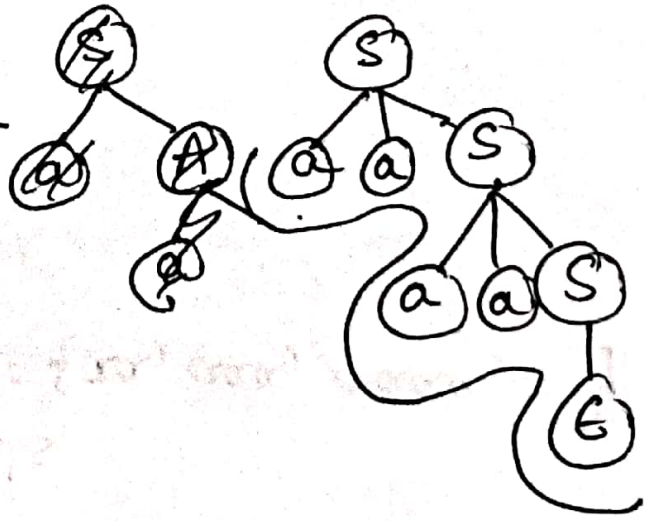
verified

or

another sol^n

$S \to aaS \mid \epsilon$

string: aaaa



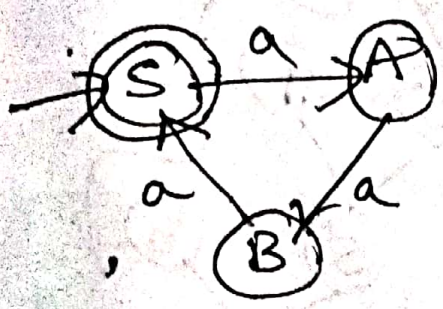
verified

6] Obtain grammar to generate string consisting of multiple of three a's

Sol^n :- $L = \{ \epsilon, aaa, aaaaaa, \dots \}$

or

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



$\delta(S,a) = A$ $\quad S \to aA$

$\delta(A,a) = B$ $\quad A \to aB$

$\delta(B,a) = S$ $\quad B \to aS$

S is a final state $\quad S \to \epsilon$

$Q = (V, T, P, S)$

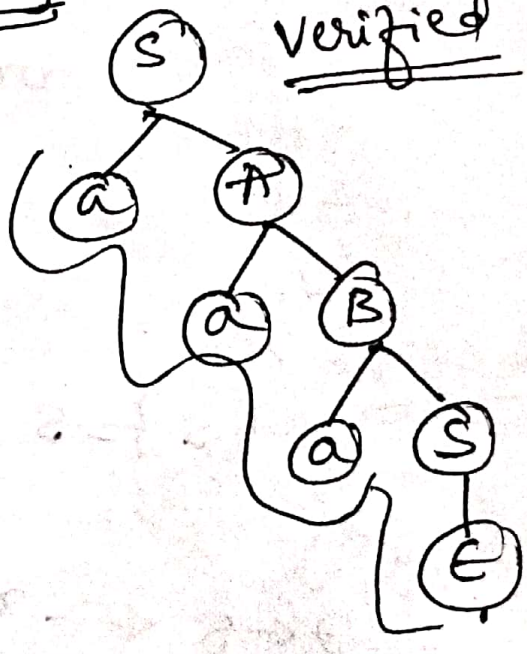
$V = \{ S, A, B \}$

$T = \{ a \}$

$P = \{ S \to aA,$
$A \to aB,$
$B \to aS,$
$S \to \epsilon \}$

$S = \{ S \}$

string: aaa

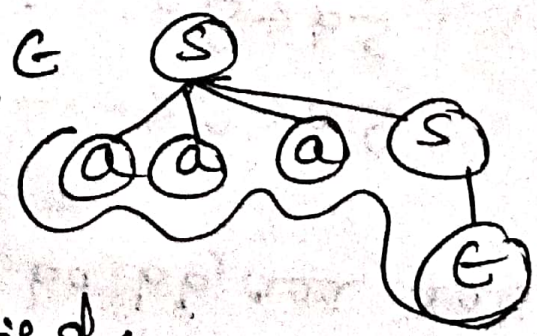


verified

another sol^n -

$S \to aaaS \mid \epsilon$

string: aaa



verified

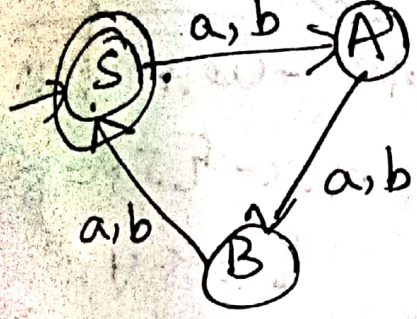
7] Obtain a grammar to generate strings of a's and b's such that string length is multiple of 3.

Sol:- $\Sigma = \{a, b\}$

$L = \{\in, aaa, aba, aab, abb, bbb, \ldots\}$.

or

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



$\delta(S,a) = A$
$\delta(S,b) = A$
$\delta(A,a) = B$
$\delta(A,b) = B$
$\delta(B,a) = S$
$\delta(B,b) = S$
S is a final state

$S \to aA$
$S \to bA$
$A \to aB$
$A \to bB$
$B \to aS$
$B \to bS$
$S \to \in$.

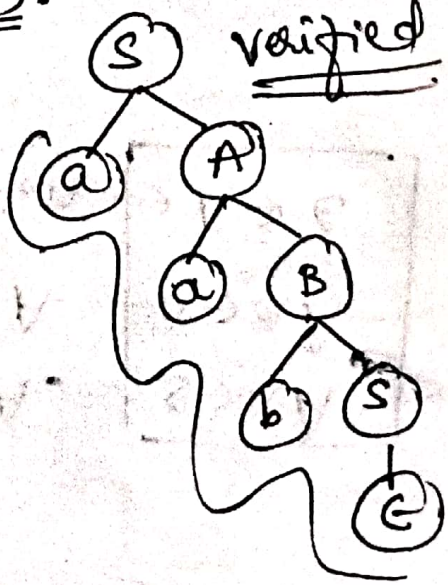
$G = (V, T, P, S)$

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

$P = \{S \to aA,\ A \to bB,\ S \to bA,\ B \to bS,\ S \to \in,\ A \to aB,\ B \to bS,\}$

$S = \{S\}$

String:- aab



verified

another sol:-

$S \to AAAS \mid \in$

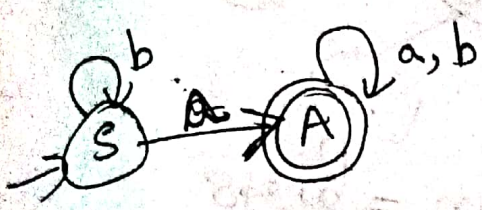
$A \to a \mid b$

8] obtain grammar to generate string consisting of any number of a's and b's with atleast one a.

Sol:- string of a's and b's with atleast one a.

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

$L = \{a, ab, ba, aab, aaab, aba, abb, \ldots\}$



$\delta(S,a) = A$
$\delta(S,b) = S$
$\delta(A,a) = A$
$\delta(A,b) = A$
A is a final state

$S \to aA$
$S \to bS$
$A \to aA$
$A \to bA$
$A \to \epsilon$

$S \to aA \}$
$S \to bS \}$
$A \to aA \}$
$A \to bA$
$A \to \epsilon$

$\Rightarrow$

$S \to aA \mid bS$
$A \to aA \mid bA \mid \epsilon$

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

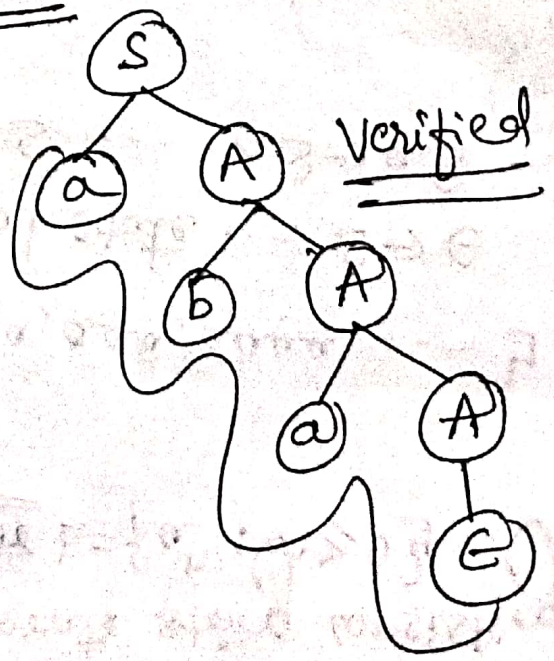
$V = \{S, A\}$

$T = \{a, b\}$

$P = \{ S \to aA \mid bS,$
$A \to aA \mid bA \mid \epsilon \}$

$S = \{S\}$

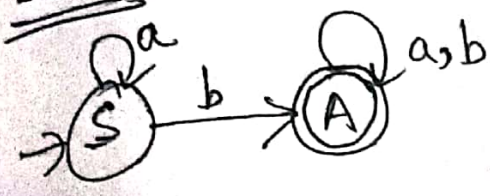
String: aba



Verified

9] obtain a grammar to generate string consisting of any number of a's and bs with at least one b.

Sol^n :- $L = \{b, ab, ba, abb, bab, \dots\}$



$\delta(S,a) = S$ $\quad S \to aS$

$\delta(S,b) = A$ $\quad S \to bA$

$\delta(A,a) = A$ $\quad A \to aA$

$\delta(A,b) = A$ $\quad A \to bA$

A is a final state $\quad A \to \in$

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

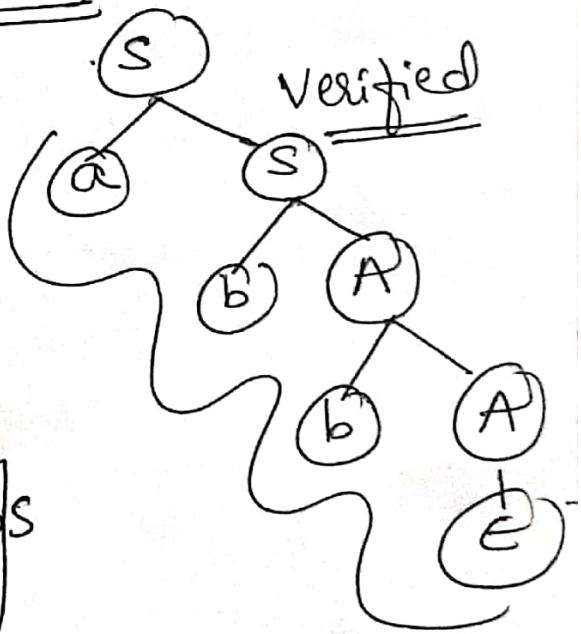
$V = \{S, A\}$

$T = \{a, b\}$

$P = \{S \to aS \mid bA,$
$A \to aA \mid bA \mid \in\}$

$S = \{S\}$

String : abb



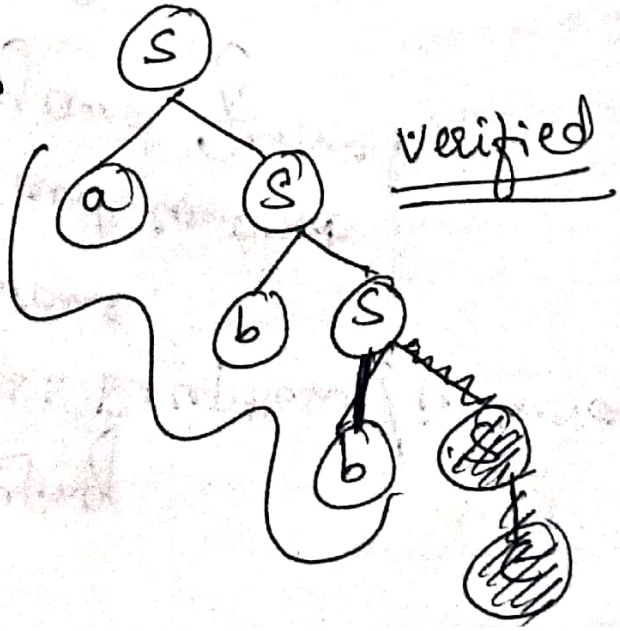
verified

another Sol^n

$S \to b$
$S \to aS$
$S \to bS$

$\Rightarrow$ $S \to b \mid aS \mid bS$

String: abb

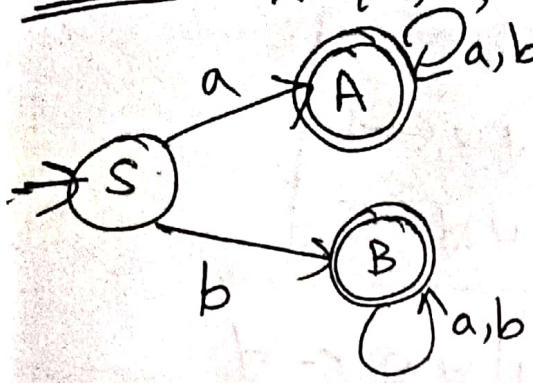


verified

10) Obtain grammar to generate string consisting of any number of a's and b's with atleast one a or atleast one b.

Sol^n

$L = \{a, b, ab, ba, abb, \ldots\}$



$\delta(S,a) = A \quad S \to aA$

$\delta(S,b) = B \quad S \to bB$

$\delta(A,a) = A \quad A \to aA$

$\delta(A,b) = A \quad A \to bA$

$\delta(B,a) = B \quad B \to aB$

$\delta(B,b) = B \quad B \to bB$

A is a final state $A \to \epsilon$

B is a final state $B \to \epsilon$

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

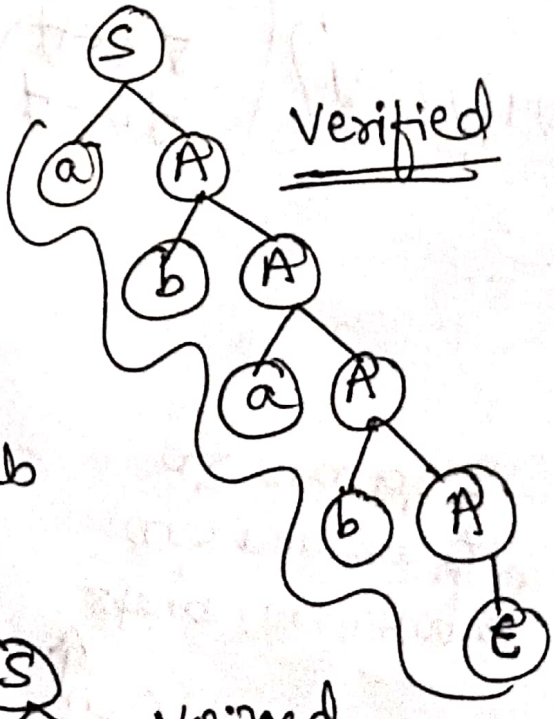
$V = \{S, A, B\}$

$T = \{a, b\}$

$P: \{ S \to aA \mid bB,$
$A \to aA \mid bA \mid \epsilon,$
$B \to aB \mid bB \mid \epsilon \}$

$S = \{S\}$

String: abab



Verified

OR

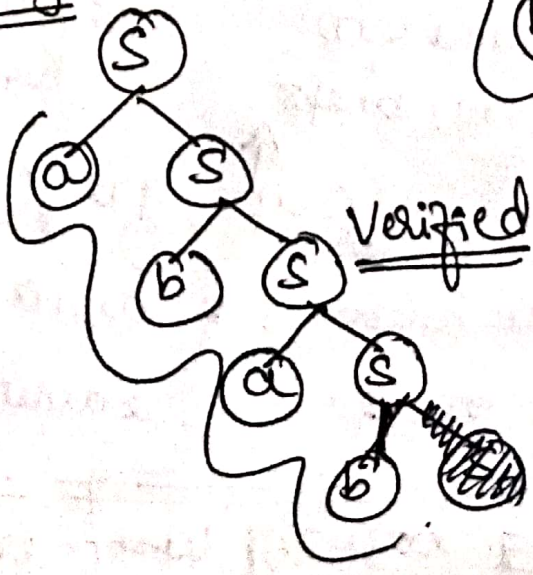
another Sol^n

$S \to a \mid b$

$S \to aS$

$S \to bS$

String: abab

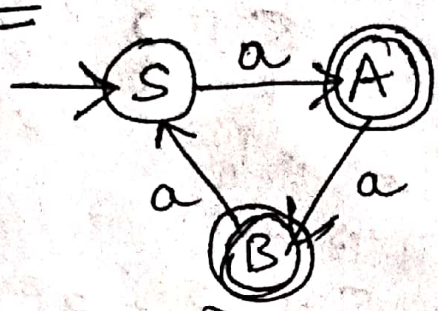


Verified

11] Obtain grammar for the following language:

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

Soln -



$\delta(S,a)=A$ $S \to aA$

$\delta(A,a)=B$ $A \to aB$

$\delta(B,a)=S$ $B \to aS$

A is a final state $A \to \epsilon$

B is a final state $B \to \epsilon$

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

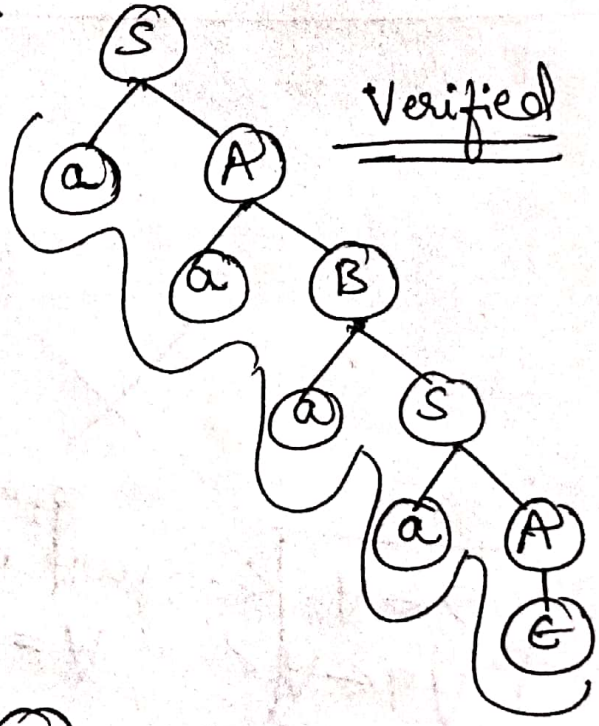
$V=\{S,A,B\}$

$T=\{a\}$

$P=\{ S \to aA,$
$A \to aB \mid \epsilon,$
$B \to aS \mid \epsilon \}$

$S=\{S\}$

String: aaaa



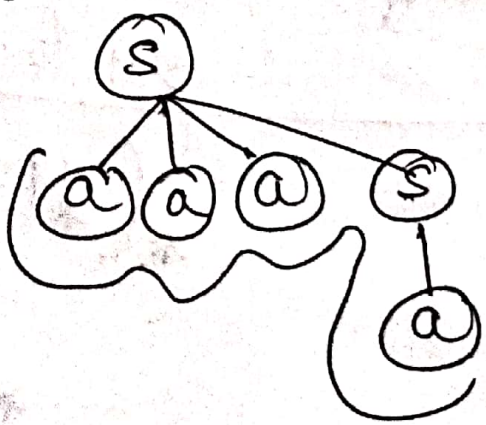
Verified

OR

another solution -

$S \to a \mid aa \mid aaaS$

String: aaaa



Verified

Grammar from Regular Expressions

12] obtain grammar to generate string of a's and b's having a substring ab.

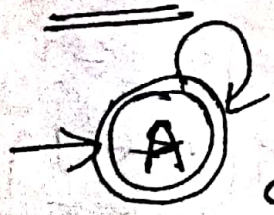
Soln - $(a+b)^* ab (a+b)^*$

$(a+b)^*$ ↓ string consisting of any number of a's and b's

$(a+b)^*$ ↓ string consisting of any number of a's and b's

↓

DFA



$\delta(A, a) = A$

$\delta(A, b) = A$

A is a final state

$A \to aA$

$A \to bA$

$A \to \epsilon$

$A \to aA \mid bA \mid \epsilon$

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

$S \to A \quad ab \quad A$

$A \to aA \mid bA \mid \epsilon$

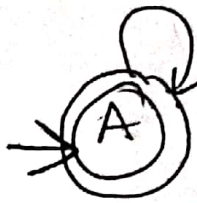
$S \to AabA$

$A \to aA \mid bA \mid \epsilon$

13] Obtain grammar to generate string of a's and b's ending with string ab.

Soln :- $(a+b)^* ab$

String consisting of any number of a's and b's



$\delta(A,a) = A$

$\delta(A,b) = A$

A is a final state

$(a+b)^* ab$

$S \rightarrow A\ ab$

$A \rightarrow aA \mid bA \mid \epsilon$

$A \rightarrow aA$, $A \rightarrow bA$, $A \rightarrow \epsilon$ $\Rightarrow A \rightarrow aA \mid bA \mid \epsilon$

14] Obtain grammar to generate strings of a's and b's starting with ab.

Soln - $ab(a+b)^*$

$S \rightarrow ab\ A$

$A \rightarrow aA \mid bA \mid \epsilon$

$\delta(A,a) = A$

$\delta(A,b) = A$

A is a final state

$A \rightarrow aA$

$A \rightarrow bA$

$A \rightarrow \epsilon$

$\Leftarrow A \rightarrow aA \mid bA \mid \epsilon$

15] obtain grammar to generate the following language:

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

Soln -

$(aa)^*$

$\Downarrow$

$(b^* a b^* a b^*)^*$.

b* → any no. of b's

$\delta(s,b) = s$

s is a final state

$(b^*ab^*ab^*)^*$

$S \to SabSaS$

$S \to bS \mid \epsilon$

$S \to bS$

$S \to \epsilon$

$S \to SaSaS$

$S \to bS \mid \epsilon$

Grammars for other Languages

16] Obtain a grammar to generate the following language:

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

Soln - $L = \{\epsilon, ab, aabb, aaabbb, \ldots\}$

check the pattern

$S \rightarrow aSb$

$S \rightarrow \epsilon$ ← minimal string.

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

$V = \{S\}$

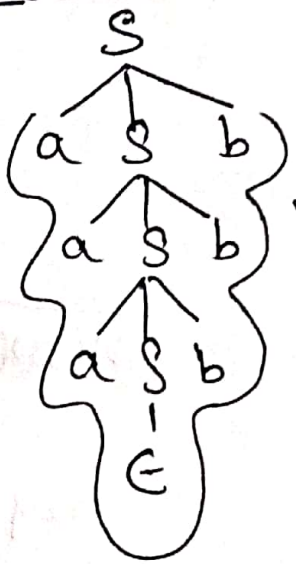
$T = \{a, b\}$

$P = \{S \rightarrow aSb \mid \epsilon\}$

$S = \{S\}$

Parse tree:

String: aaabbb



verified

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

Soln: $L = \{ab, aabb, aaabbb, \ldots\}$

$S \rightarrow aSb$

$S \rightarrow ab$ ← combine $\Rightarrow$ $S \rightarrow aSb \mid ab$.

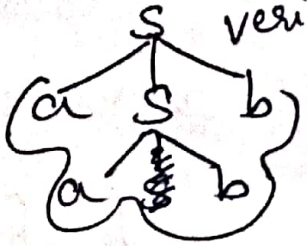
$G = (V, T, P, S)$

$V = \{S\}$ $P = \{S \rightarrow aSb \mid ab\}$

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

Parse tree

aabb



verified

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

Soln $L = \{ab, aab, aaabb, \ldots\}$

$L = \{a, \underline{a}ab, \underline{a}aab\underline{b}, \ldots\}$

$S \to aSb$
$S \to a$

$S \to asb \mid a$

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

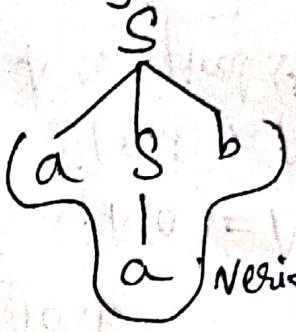
$V = \{S\}$ $S = \{S\}$

$T = \{a, b\}$

$P = \{S \to aSb \mid a\}$

Parse tree:

String - aab



Verified.

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

Soln $L = \{b, \underline{a}ab\underline{b}, \underline{a}abbb\underline{b}, \ldots\}$

$S \to aSb$
$S \to b$

$S \to asb \mid b$

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

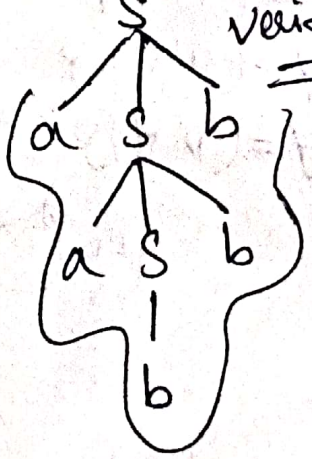
$V = \{S\}$ $S = \{S\}$

$T = \{a, b\}$

$P = \{S \to aSb \mid b\}$

Parse tree:

String: aabbb



Verified.

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

Soln: $L = \{bb, abbb, aabbbb, \ldots\}$

$S \to aSb$
$S \to bb$

$S \to aSb \mid bb$

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

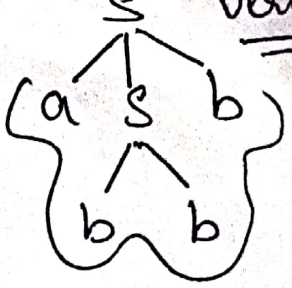
$V = \{S\}$ $S = \{S\}$

$T = \{a, b\}$

$P = \{S \to aSb \mid bb\}$

Parse-tree:

String: abbb



verified.

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

Soln: $L = \{\epsilon, abb, aabbbb, aaabbbbbb, \ldots\}$

$S \to aSbb$
$S \to \epsilon$

$S \to aSbb \mid \epsilon$

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

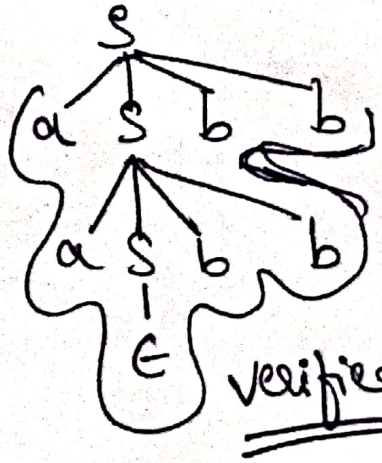
$V = \{S\}$ $S = \{S\}$

$T = \{a, b\}$

$P = \{S \to aSbb \mid \epsilon\}$

Parse tree-

String: aabbbb



verified

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

Sol^n - $L = \{\epsilon, aa, bb, abba, baab, aaaa, \cdots\}$

$L = \{\epsilon, \underline{a}a\underline{a}, \underline{b}b\underline{b}, \underline{a}bb\underline{a}, \underline{b}aa\underline{b}, \underline{a}aa\underline{a}, \cdots\}$

$S \to aSa$
$S \to bSb$
$S \to \epsilon$

$S \to aSa \mid bSb \mid \epsilon$

parse tree - string abba

verified

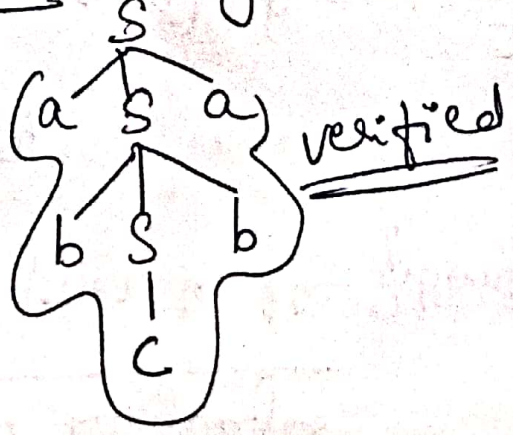
2) $L = \{ wcw^R \text{ where } w \in \{a,b\}^* \}$

Sol^n - $L = \{c, aca, bcb, abcba, bacab, \cdots\}$

$S \to aSa$
$S \to bSb$
$S \to c$

$S \to aSa \mid bSb \mid c$

parse tree - string abcba



verified

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

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

$L = A\,B$

$S \to AB$

$A \to 0^m 1^m$: $L = \{01, 0011, \ldots\}$

$A \to 0A1$

$A \to 01$

$B \to 2^n$: $L = \{\epsilon, 2, 22, \ldots\}$

$B \to 2B$

$B \to \epsilon$

$S \to AB$
$A \to 0A1$
$A \to 01$
$B \to 2B$
$B \to \epsilon$

$\Longrightarrow$

$S \to AB$
$A \to 0A1 \mid 01$
$B \to 2B \mid \epsilon$

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

Sol^n - $L = \{\epsilon, ab, ba, aabb, bbaa, abba, \ldots\}$

Three cases -

(i) An empty string denoted by ϵ has equal number of a's and b's.

(ii) The symbol 'a' can be followed by the symbol 'b'.

(iii) The symbol 'b' can be followed by the symbol 'a'.

$S \to \epsilon$

$S \to aSb$

$S \to bSa$

abba
baab, "Starts and ends with the same symbol.

The production →

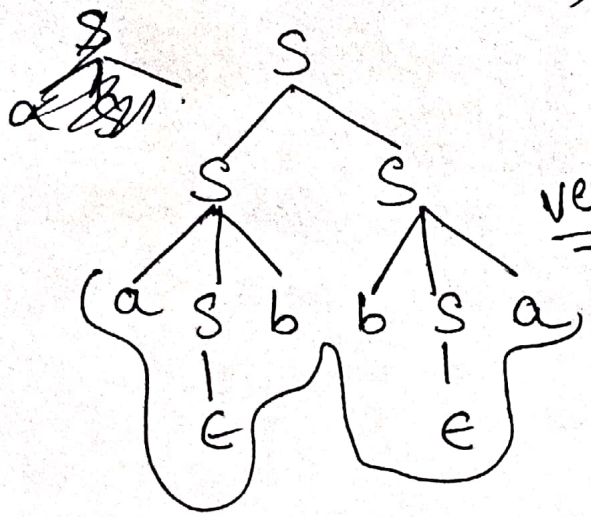
$S \to SS$

Final grammar -

$S \to \epsilon \mid asb \mid bSa \mid SS$

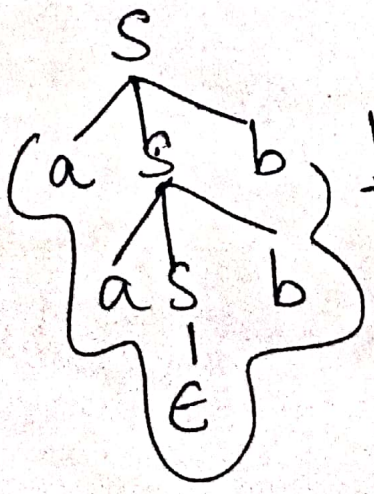
Parse tree - abba

a=2, b=2



verified.

aabb



verified

//.

7.5 Derivation Tree (Parse Tree)

The derivation can be shown in the form of a tree. Such trees are called derivation or parse trees. The leftmost derivation as well as the right most derivation can be represented using derivation trees. Now, let us see **"What is derivation tree or parse tree?"**

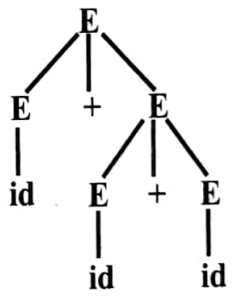
Definition (Parse tree or Derivation Tree): Let G = (V, T, P, S) be a context-free grammar. The tree is derivation tree (parse tree) is defined with the following properties.

- The root has the label S.
- Every vertex has a label which is in (V U T U ε).
- Every leaf node has label from T and an interior vertex has a label from V.
- 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.

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

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

Derivation tree

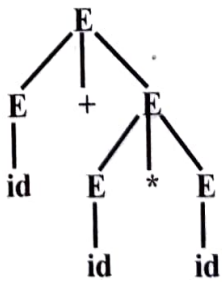


Note that at each step in the derivation process, only the left most variable E is replaced and so the derivation is said to be leftmost.]

Rightmost derivation: The rightmost derivation to get the string id + id * id is shown below:

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

Derivation tree



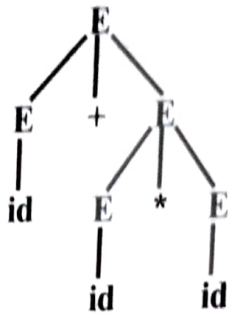
Note: In the above derivation tree, let us read only the leaf nodes from left to right (leaving the ε-symbols if any). The string obtained:

id + id * id

is called the ***yield*** of the tree. Now, let us see **"What is the yield of the tree?"** The *yield* of a tree can be formally defined as follows:

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

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



If we read only the terminal symbols in the above tree from left to right we get:

id + id * id

and **id + id * id** is the yield of the given parse tree.

7.5.1 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: We can check whether the grammar is ambiguous or not as shown below:

- 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.
- 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.
- 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.
- 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 7.38:

Consider the following grammar:

$E \rightarrow E + E \mid E - E$

$E \rightarrow E * E \mid E / E \mid (E) \mid \mathbf{id}$

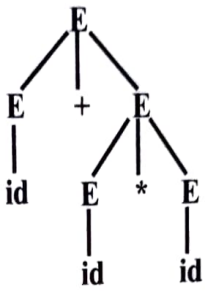
Show that the grammar is ambiguous.

Solution:

The string **id + id * id** can be obtained by applying leftmost derivation as shown below. The derivation tree for the leftmost derivation is also shown below:

Leftmost derivation **Derivation tree**

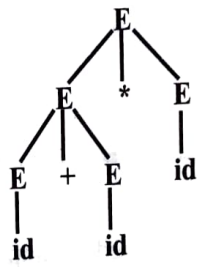
$E \Rightarrow E + E$
$\Rightarrow$ **id** + E
$\Rightarrow$ **id** + E * E
$\Rightarrow$ **id** + **id** * E
$\Rightarrow$ **id** + **id** * **id**



The same string **id + id * id** is obtained by applying leftmost derivation but using different sets of productions as shown below. The corresponding derivation tree is also shown below:

Leftmost derivation **Derivation tree**

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



Since there are two different parse trees for the same string **id + id * id,** the given grammar is ambiguous.

Example 7.39:

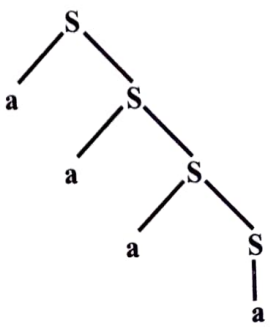
Is the following grammar ambiguous?

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

Solution:

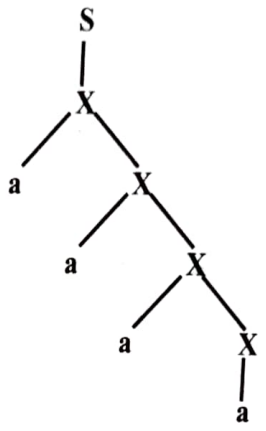
The **leftmost derivation** for the string **aaaa** along with equivalent parse tree can be written as shown below:

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



The same string **aaaa** can be obtained by **applying leftmost derivation** but using different sets of productions. The leftmost derivation for the string **aaaa** along with parse tree is shown below:

$$
\begin{aligned}
S &\Rightarrow X \\
&\Rightarrow aX \\
&\Rightarrow aaX \\
&\Rightarrow aaaX \\
&\Rightarrow aaaa
\end{aligned}
$$



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

Example 7.40: Is the following grammar ambiguous?

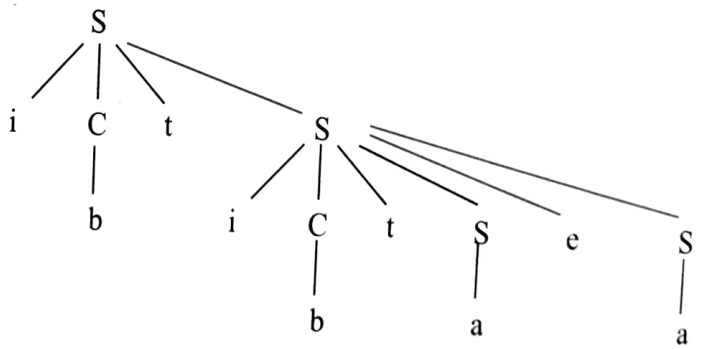
$S \rightarrow iCtS \mid iCtSeS \mid a$

$C \rightarrow b$

Solution:

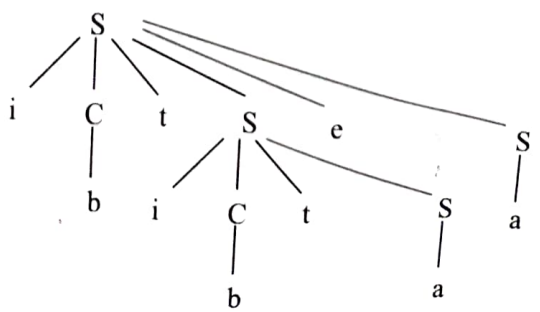
The string **ibtibtaea** can be obtained by applying the leftmost derivation as shown below:

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



The same string **ibtibtaea** can be obtained again by applying leftmost derivation and equivalent parse tree can be written as shown below:

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



7.6.4 Simplification of CFG

In this section, let us see how simplify a CFG, by eliminating useless symbols and productions. Let G = (V, T, P, S) be a CFG. In the grammar G, two types of symbols may be present:

- some of the symbols or productions may not be used to derive a string
- 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 the above grammar, if we apply the production $S \rightarrow B$, **from non-terminal B, a string can never be derived. So, the symbol B and the production $S \rightarrow B$ 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.

Theorem 7.1:

Let G = (V, T, P, S) be a CFG. We can find an equivalent context-free grammar:

$$G' = (V' , T' , P' , S)$$

such that for each A in (V' U T') there exists α and β in (V' U T')* and x in T^+ for which

$$S \overset{*}{\Rightarrow} \alpha A \beta \overset{*}{\Rightarrow} x.$$

Example 7.50: Eliminate useless symbols in the following grammar

$S \rightarrow aA \mid bB$

$A \rightarrow aA \mid a$

$B \rightarrow bB$

$D \rightarrow ab \mid Ea$

$E \rightarrow ac \mid d$

Solution:

The useless symbols can be eliminated using two states:

Stage 1: Find the set of symbols and productions from which we get only string of terminals. This can be done using the algorithm:

$ov = \phi$

$nv = ov \cup \{A \mid A \rightarrow y \text{ and } y \in (ov \cup T)^* \}$

while (ov ! = nv)

$ov = nv;$

$nv = ov \cup \{A \mid A \rightarrow y \text{ and } y \in (ov \cup T)^* \}$

end while

$V_1 = ov$

Oldv (ov)	Newv (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 → a | aA
D → ab | Ea
E → d
S → aA

}

S is the start symbol

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

Stage 2: Using the above grammar we obtain the symbols such that each symbol X is reachable from the start symbol S as shown below using the following algorithm:

V' = {S}
for each A in V'
  if A → α then
    Add the variables in α to V'
    Add the terminals in α to T'
  end if
end for

P'	T'	V'
–	-	S
S → aA	a	S, A
A → a | aA	a	S, A

The resulting grammar G' = (V', T', P', S) where

V' = {S, A}
T' = {a}
P' = {
  S → aA
  A → a | aA
}
S is the start symbol

such that each symbol X in (V' U T') has a derivation of the form

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

Example 7.51: Simplify the following grammar

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

Solution: The useless symbols can be eliminated using two states:

Stage 1: **Find the set of symbols and productions from which we get only string of terminals. This can be done using the algorithm:**

ov = ϕ

nv = ov U {A | A → y and y ∈ (ov U T)* }

while (ov ! = nv)

ov = nv;

nv = ov U {A | A → y and y ∈ (ov U T)* }

end while

V_1 = ov

Oldv (ov)	Newv (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

D → ddd

A → aB

}

S is the start symbol

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

Stage 2: Using the above grammar we obtain the symbols such that each symbol X is reachable from the start symbol S as shown below using the following algorithm:

V′ = {S}

for each A in V′

if A → α then

Add the variables in α to V′

Add the terminals in α to T'

end if

end for

P'	T'	V'
–	-	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' = (V' , T' , P' , S) where

V' = { S, A, B }

T' = {a, b}

P' = {

S → a | Bb | aA

A → aB

B → a | Aa

}

S is the start symbol

such that each symbol X in (V' U T') has a derivation of the form

$$S \stackrel{*}{\Rightarrow} \alpha X \beta \stackrel{*}{\Rightarrow} w$$

7.6.5 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. Now, let us see **"What is an e-production? What is nullable variable?"**

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

$$A \rightarrow \varepsilon$$

is called an **ε-production.** It is also called **NULL production**. If we apply the production: A → ε, then in the derivation, the variable A is replaced by ε (i.e., the variable A is erased.) For each A in V, if there is a derivation of the form

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

then A is 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.

- If A → ε is a production in P, then A is a **nullable variable.**

- If $A \rightarrow B_1B_2...B_n$ is a production in P, and if $B_1, B_2, ...B_n$ are nullable variables, then A is also a **nullable variable**
- The variables for which there are productions of the form shown in previous two steps are **nullable variables.**

For example, consider the following grammar:

$$S \rightarrow ABCa \mid bD$$

$$A \rightarrow BC \mid b$$

$$B \rightarrow b \mid \varepsilon$$

$$C \rightarrow c \mid \varepsilon$$

$$D \rightarrow d$$

In the above grammar, the productions:

$$\mathbf{B \rightarrow \varepsilon \text{ and } C \rightarrow \varepsilon}$$

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

$$A \rightarrow BC$$

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

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. Now, let us see **"How to eliminate ε-productions?"**

Example 7.52: Eliminate ε-productions from the following grammar:

$S \rightarrow ABCa \mid bD$

$A \rightarrow BC \mid b$

$B \rightarrow b \mid \varepsilon$

$C \rightarrow c \mid \varepsilon$

$D \rightarrow d$

Solution:

Step 1: Obtain the set of nullable variables from the grammar. This can be done using the following algorithm:

step

ov = ϕ

nv = { A | A → ε }

while (ov != nv)

ov = nv

nv = ov ∪ { A | A → α and α ∈ ov* }

end while

V = ov

Oldv (ov)	Newv (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.

Step 2: Construction of productions P'. Take all combinations of nullable variables, delete one by one and add the resulting productions to P'.

Productions	Resulting productions (P')
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

The final grammar obtained after removing e-productions is shown below:

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

Example 7.53: Eliminate all ε- productions from the grammar

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

Solution:

Step1: Obtain the set of nullable variables from the grammar. This can be done using step the following algorithm:

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

Oldv (ov)	Newv (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.

Step 2: Construction of productions P'. Add a non $\in$-production in P to P'. 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'.

Productions	Resulting productions (P')
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

We can delete the productions of the form A → A. In P', the production B → B can be deleted and the final grammar obtained after eliminating ε-productions is shown below.

S → BAAB | AAB | BAB | BAA | AB | BB | BA | AA | A | B

A → 0A2 | 02 | 2A0 | 20

B → AB | A | 1B | 1

7.6.6 Eliminating Unit Productions

In this section, let us eliminate unit productions. First, let us see **"What is an unit production?"**

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

A → B

where A, B ∈ V is a **unit-production.** In any grammar, the unit productions are undesirable. This is because one variable is simply replaced by another variable.

For example, consider the following grammar:

A → B

B → aB | b

In the above grammar, the production:

A → B

is **unit-production** and the productions:

B → aB

B → b

are **non-unit productions.**

Now, let us see **"How to eliminate unit productions?"** The unit productions can be eliminated using the theorem shown below:

Theorem 7.3:

Let G = (V, T, P, S) be a CFG and has unit productions and no ε-productions. An equivalent grammar G_1 without unit productions can be obtained such that $L(G) = L(G_1)$

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

- Remove all the productions of the form $A \rightarrow A$
- Add all non-unit productions to P_1.
- 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.

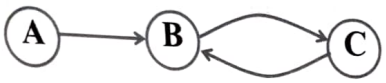
- Obtain a dependency graph. For example, if we have the productions

$A \rightarrow B$

$B \rightarrow C$

$C \rightarrow B$

the dependency graph will be of the form



- 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.

- Finally, the unit productions can be deleted from the grammar G.
- The resulting grammar G_1, generates the same language as accepted by G.

Example 7.54: Eliminate all unit productions from the grammar

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow C \mid b$

$C \rightarrow D$

$D \rightarrow E \mid bC$

$E \rightarrow d \mid Ab$

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

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow b$

$D \rightarrow bC$

$E \rightarrow d \mid Ab$

The unit productions of the grammar G are shown below: ...7.1

$B \rightarrow C$

$C \rightarrow D$

$D \rightarrow E$

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

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

  So, $E \rightarrow d \mid Ab \quad \Rightarrow \quad D \rightarrow d \mid Ab$

  Now, the resulting D productions are:

  $D \rightarrow bC$

  $D \rightarrow d \mid Ab$ ...7.2

- It is clear from the dependency graph that, $C \stackrel{*}{\Rightarrow} E$. So, the non-unit productions from E can be generated from C. So, $C \rightarrow d \mid Ab$ ...7.3

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

  $C \rightarrow bC$

  $C \rightarrow d \mid Ab$ ...7.4

- From the dependency graph it is clear that $B \stackrel{*}{\Rightarrow} C$, $B \stackrel{*}{\Rightarrow} D$, $D \stackrel{*}{\Rightarrow} E$. So, all the productions obtained from B can be obtained using (7.1), (7.2) (7.3) and (7.4) and the resulting productions are:

  $B \rightarrow b$

  $B \rightarrow d \mid Ab$

  $B \rightarrow bC$ ...7.5

The **final grammar obtained after eliminating unit productions** can be obtained by combining the productions (7.1), (7.2), (7.3), (7.4) and (7.5) and is shown below:

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow b \mid d \mid Ab \mid bC$

$C \rightarrow bC \mid d \mid Ab$

D → bC | d | Ab

E → d | Ab

Example 7.55: Eliminate unit productions from the grammar

S → A0 | B

B → A | 11

A → 0 | 12 | B

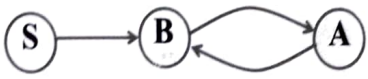
Solution: The dependency graph for the unit productions:

S → B

B → A

A → B

is shown below:



The non-unit productions are:

S → A0 ...7.6

B → 11

A → 0 | 12

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:

S → 11 | 0 | 12 ...7.7

B → 0 | 12

A → 11

The resulting grammar without unit productions can be obtained by combining (7.6) and (7.7) and is shown below:

S → A0 | 11 | 0 | 12

A → 0 | 12 | 11

B → 11 | 0 | 12

Example 7.56: Eliminate unit productions from the grammar

S → Aa | B | Ca

B → aB | b

C → Db | D

D → E | d

E → ab

Solution: The dependency graph for the unit productions:

$S \to B$

$C \to D$

$D \to E$

is shown below.

The non-unit productions are:

$S \to Aa \mid Ca$

$B \to aB \mid b$

$C \to Db$ ...7.8

$D \to d$

$E \to ab$

- 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:

  $S \to aB \mid b$ ...7.9

- 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:

  $D \to ab \mid d$ ...7.10

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

  $C \to Db \mid ab \mid d$ ...7.11

  The resulting grammar without unit productions can be obtained by combining (7.8), (7.9), (7.10) and (7.11) and is shown below:

  $S \to Aa \mid Ca \mid aB \mid b$

  $B \to aB \mid b$

  $B \to Db \mid ab \mid d$

  $B \to d \mid ab$

  $B \to ab$

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

- ε - productions using theorem 7.2
- unit productions using theorem 7.3
- useless symbols and productions using theorem 7.1 in above sequence. The final grammar obtained does not have any undesirable productions.

7.7 Normal Forms

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:

- Chomsky Normal Form (CNF)
- Greiback Normal Form (GNF)

7.7.1 Chomsky Normal Form (CNF)

In this section, let us see **"What is Chomsky Normal Form?"**

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 not symbol.

- If there are two symbols on the right-hand side those two symbols must be non-terminals.
- If there is only one symbol, that symbol must be a terminal.

Now, let us see **"How to convert a given grammar into CNF?"** This can be explained using the theorem shown below:

Theorem 7.5: 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' = (V', T, P', S) in CNF such that L(G) = L(G'). That is, all productions in G' 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' can be obtained using the following steps.

Step 1: 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, $L(G) = L(G_1)$.

Step 2: 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'. 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* cannot be equal to 1. Because if n = 1. Because if n = 1, there is only one symbol and it is terminal which again is in CNF). The A-production can be written as

$$A \rightarrow A_1D_1$$
$$D_1 \rightarrow A_2D_2$$
$$D_3 \rightarrow A_3D_3$$

.

.

.

$$D_{n-2} \rightarrow A_{n-1}A_n$$

These productions are added to P' and new variables are added to V'. The grammar thus obtained is in CNF. The resulting grammar G' = (V', T, P', S) generates the same language as accepted by G i.e. L(G) = L(G').

Example 7.57:

Convert the following grammar into CNF

$S \rightarrow 0A \mid 1B$

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

$B \rightarrow 1BB \mid 0S \mid 0$

Solution:

- All productions which are in CNF are added to P_1. The productions which are in standard form and added to P1 are:

$$A \rightarrow 1$$
$$B \rightarrow 0 \qquad \text{...7.12}$$

- **Consider all productions which are not in CNF. If RHS of any production contains a terminal *a*, replace the terminal *a* on right hand side of the production by a non-terminal A and introduce the production A → a.** This step has to be carried out for each production which are not in CNF as shown below:

Given Production	Action	Resulting production
$S \rightarrow 0A \mid 1B$	Replace 0 by B_0 and add $B_0 \rightarrow 0$ Replace 1 by B_1 and add $B_1 \rightarrow 1$	$S \rightarrow B_0A \mid B_1B$ $B_0 \rightarrow 0$ $B_1 \rightarrow 1$
$A \rightarrow 0AA/1S$	Replace 1 by B_1 and add $B_1 \rightarrow 1$ Replace 1 by B_1 and add $B_1 \rightarrow 1$	$A \rightarrow B_0AA \mid B_1S$
$B \rightarrow 1\ BB/0S$	Replace 0 by B_0 and add $B_0 \rightarrow 0$ Replace 1 by B_1 and add $B_1 \rightarrow 1$	$B \rightarrow B_1BB \mid B_0S$

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 (7.12).

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

$T_1 = \{0, 1\}$

$P_1 = \{$

$S \rightarrow B_0A \mid B_1B$

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

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

$B_0 \rightarrow 0$

$B_1 \rightarrow 1$

$\}$

S is the start symbol

Step 2: Restricting the number of variables on the right-hand side of the production to 2. In the above productions, the productions which are in CNF are shown below:

$$\begin{aligned} S &\rightarrow B_0A \mid B_1B \\ A &\rightarrow B_1S \mid 1 \\ B &\rightarrow B_0S \mid 0 \\ B_0 &\rightarrow 0 \\ B_1 &\rightarrow 1 \end{aligned} \quad \ldots 7.13$$

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

$A \rightarrow B_0AA$

$B \rightarrow B_1BB$

Now, let us restrict the number of variables on the right-hand side to 2 as shown below:

Given Production	Action	Resulting production
$A \rightarrow B_0AA$	Replacing AA by D1	$S \rightarrow B_0D_1$ $D_1 \rightarrow AA$
$B \rightarrow B_1BB$	Replacing BB by D2	$S \rightarrow B_1D_2$ $D_2 \rightarrow BB$

The final grammar which is in CNF can be obtained by combining the productions in 7.13 and the resulting productions in above table. So, the grammar G' = (V', T, P', S) is in CNF where:

$S \rightarrow B_0A \mid B_1B$

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

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

$B_0 \rightarrow 0$

$B_1 \rightarrow 1$

$D_1 \rightarrow AA$

$D_2 \rightarrow BB$

7.7.2 Greibach Normal Form (GNF)

In CNF there is restriction on the number of symbols on the right-hand side of the production. Note that in CNF not more than two symbols on R.H.S of the production are permitted. If there is only symbol that symbol must be a terminal and if there are two symbols, those two symbols must be variables.

In GNF there is no restriction on the number of symbols on the right-hand side, but there is restriction on the terminals and variables appear on the right-hand side of the production. Now, let us see **"What is GNF? How to convert a grammar into GNF?"**

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

$$A \rightarrow a\alpha$$

where $a \in T$ and $\alpha \in V^*$. That is, the first symbol on the right-hand side of the production must be a terminal and it can be followed by zero or more variables.

For example, the following grammar is in GNF:

$S \rightarrow aAA \mid b$

$A \rightarrow bB \mid a$

$B \rightarrow bB \mid b$

Now, let us see **"How to convert a grammar into GNF?"** The procedure is shown below:

Step 1: Obtain the grammar in CNF.

Step 2: Rename the non-terminals to $A_1, A_2, A_3, \ldots\ldots$

Step 3: Using substitution method as discussed in section 7.6.1, obtain the productions to the form

$$A_i \rightarrow A_j\alpha \qquad \text{for } i < j$$

where $\alpha \in V^*$. Note if all the productions are in this manner, the number of steps will be reduced while converting.

Step 4: After substitution, if a grammar has left-recursion, we should eliminate left recursion as discussed in section 7.6.3.

Step 5: It may be necessary to apply step 3 and/or step 4 more than once to get the grammar in GNF.

Example 7.58:

Convert the following grammar into GNF

$$S \rightarrow AB1 \mid 0$$
$$A \rightarrow 00A \mid B$$
$$B \rightarrow 1A1$$

Solution:

Note: A grammar should not have any unit productions and ε-productions. In case if the grammar has ε-productions and unit productions perform the following operations one after the other:

- Eliminate all ε-productions.
- Eliminate all unit productions
- Obtain the grammar in CNF
- Finally convert the grammar into GNF.

Since, the grammar does not have any ε-productions, let us eliminate the unit production:

$$A \rightarrow B$$

The dependency graph for this can be:

It is clear from the dependency graph that $A \Rightarrow B$. So, all the symbols derivable from B area also derivable from A. So, in the production.

$$A \rightarrow 00A \mid B$$

the variable B can be replaced by the string 1A1 using the production

$$B \rightarrow 1\ A\ 1$$

Now, the A-production can be written as

$$A \rightarrow 00A \mid 1A1$$

The resulting grammar obtained after eliminating unit productions is shown below:

$$S \rightarrow AB1 \mid 0$$
$$A \rightarrow 00A \mid 1A1$$
$$B \rightarrow 1A1$$

Now, the grammar has to be converted into CNF. Now replace the terminals by non-terminals if they are not in CNF and the resulting grammar is

$$S \rightarrow ABA_1 \mid 0$$
$$A \rightarrow A_0A_0A \mid A_1AA_1$$
$$B \rightarrow A_1AA_1$$
$$A_1 \rightarrow 1$$
$$A_0 \rightarrow 0$$

Now, restrict the number of variables on the right-hand side of the production to two and the resulting grammar in CNF notation is:

$$S \rightarrow AD_1 \mid 0$$
$$A \rightarrow A_0D_2 \mid A_1D_3$$
$$B \rightarrow A_1D_3$$
$$A_1 \rightarrow 1$$
$$A_0 \rightarrow 0$$
$$D_1 \rightarrow BA_1$$
$$D_2 \rightarrow A_0A$$
$$D_3 \rightarrow AA_1$$

Now let us rename all the variables as shown below:

Let $S = A_1$, $A = A_2$, $B = A_3$, $A_0 = A_4$, $A_1 = A_5$, $D_1 = A_6$, $D_2 = A_7$, $D_3 = A_8$. Now, the grammar can be re-written as

$$A_1 \rightarrow A_2A_6 \mid 0$$
$$A_2 \rightarrow A_4A_7 \mid A_5A_8$$
$$A_3 \rightarrow A_5A_8$$
$$A_5 \rightarrow 1$$
$$A_4 \rightarrow 0$$
$$A_6 \rightarrow A_3A_5$$
$$A_7 \rightarrow A_4A_2$$
$$A_8 \rightarrow A_2A_5$$

In the above productions, note that A_4 and A_5-productions are in GNF.

Consider A_3 production: Substituting for A_5 in A_3-production we get

$$A_3 \rightarrow A_5A_8 = 1A_8$$

Now, A_3 - production is in GNF.

Consider A_2 production: Since A_4 and A_5 productions are in GNF, substituting these productions in A_2 - production we get

$$A_2 \rightarrow 0A_7 \mid 1A_8$$

Now, A_2-production is also in GNF.

Consider A_1 production: Since A_2 production is in GNF, substituting for A_2 in A1-production we get

$$A_1 \rightarrow 0A_7A_6 \mid 1A_8A_6 \mid 0$$

Now, A_1 production is also in GNF.

Consider A_6-production: Now, in A_6-production, replacing the first A_3 by A_3-production we get the following A_6-production

$$A_6 \rightarrow A_5A_8A_5$$

which is not in desired form. In the above production, replacing the first A_5 by A_5-production, we get A_6-production in GNF as shown below:

$$A_6 \rightarrow 1A_8A_5$$

Consider A_7 production: Replacing the first A_4 in A_7-production we get

$$A_7 \rightarrow 0A_2$$

which is in GNF.

Consider A_8 production: Since A_2 production is in GNF, substituting for A_2 in A8-production we get

$$A_8 \rightarrow 0A_7A_5 \mid 1A_8A_5$$

which is in GNF. Since all productions are in GNF, the whole grammar is in GNF. The final grammar which is in GNF is G = (V, T, P, S) where

$V = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8\}$

$T = \{0, 1\}$

$P = \{$

$A_1 \rightarrow 0A_7A_6 \mid 1A_8A_6 \mid 0$

$A_2 \rightarrow 0A_7 \mid 1A_8$

$A_3 \rightarrow 1A_8$

$A_4 \rightarrow 0$

$A_5 \rightarrow 1$

$A_6 \rightarrow 1A_8A_5$

$A_7 \rightarrow 0A_2$

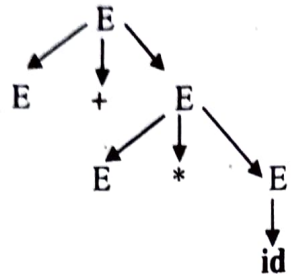
$A_8 \rightarrow 0A_7A_5 \mid 1A_8A_5$

$\}$

S is the start symbol

Note: In this problem there are only substitutions and after repeated substitutions, we get the grammar in GNF.

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



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

5.6 Ambiguous grammar

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

Note:

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

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

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

Show that the grammar is ambiguous.

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

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

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

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

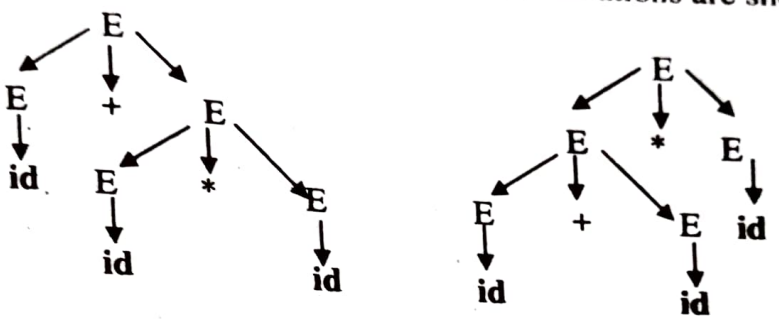


fig.5.2 Derivation trees for Leftmost derivation

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

Example 5.16: Is the following grammar ambiguous?

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

Consider the two leftmost derivations for the string aaaa.

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

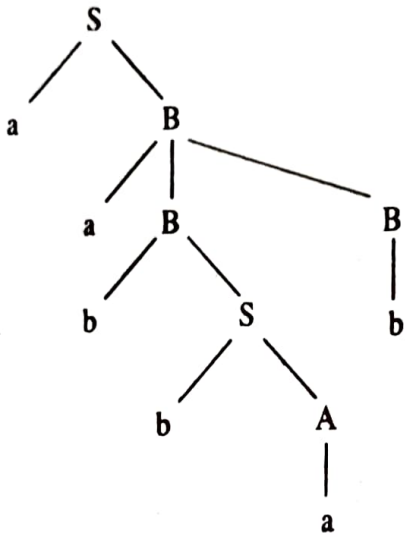
Since there are two leftmost derivations for the same sentence aaaa, the given grammar is ambiguous.

Example 5.17: Is the following grammar ambiguous?

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

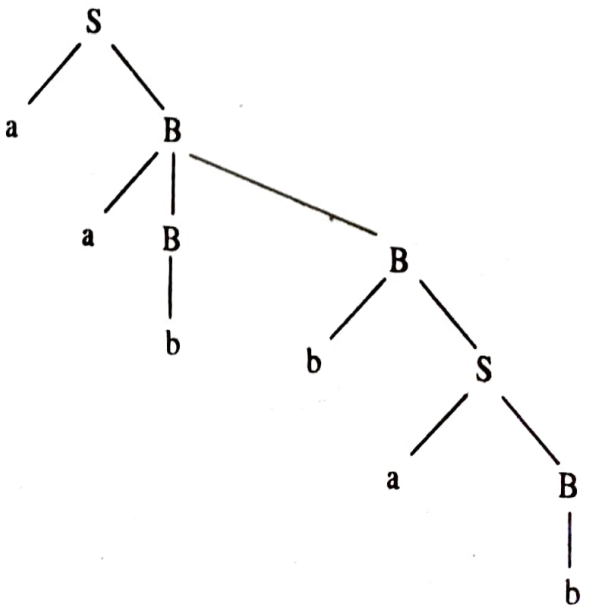
Solution: Consider the leftmost derivation and the corresponding pares tree shown below:

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



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

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



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

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

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

The leftmost derivation for the string aaabbabbba is shown below:

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

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

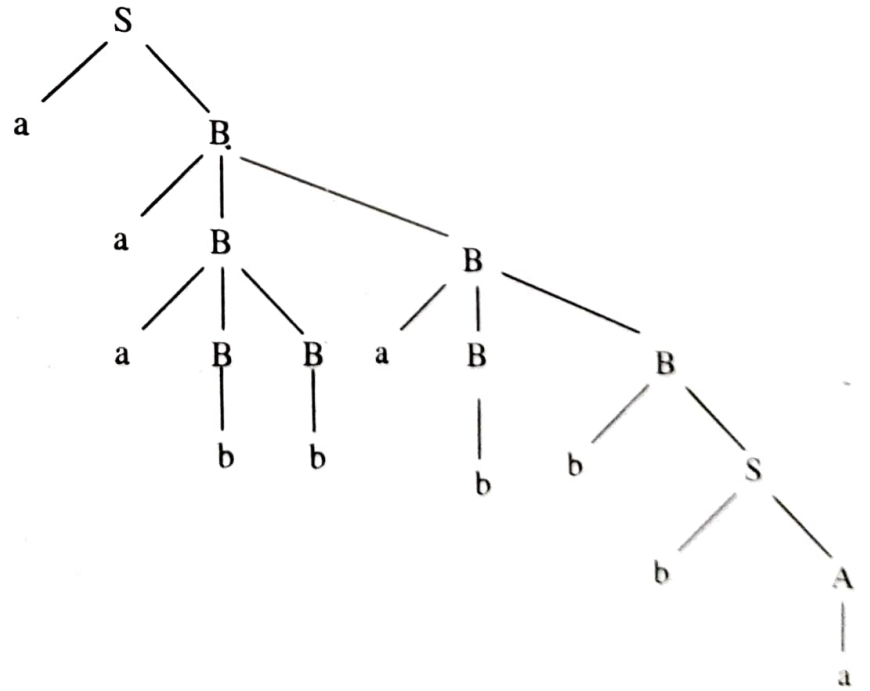


fig.5.3 Derivation tree for Leftmost derivation

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

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

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

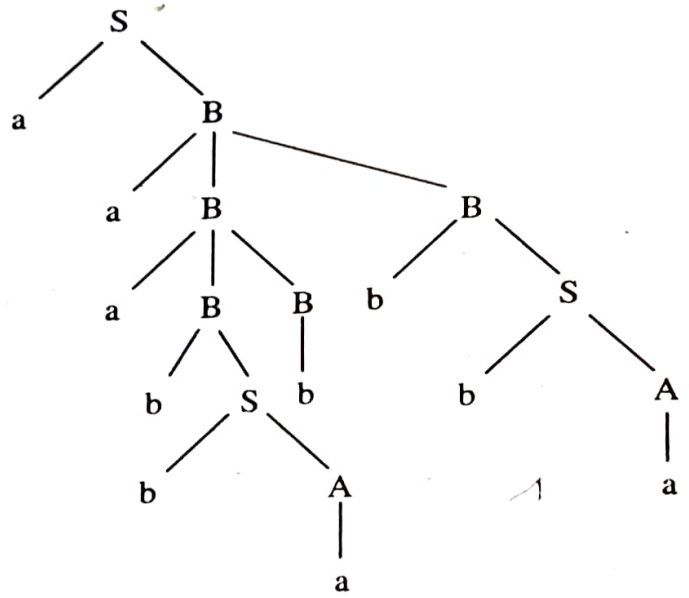


fig.5.4 Derivation tree for Leftmost derivation

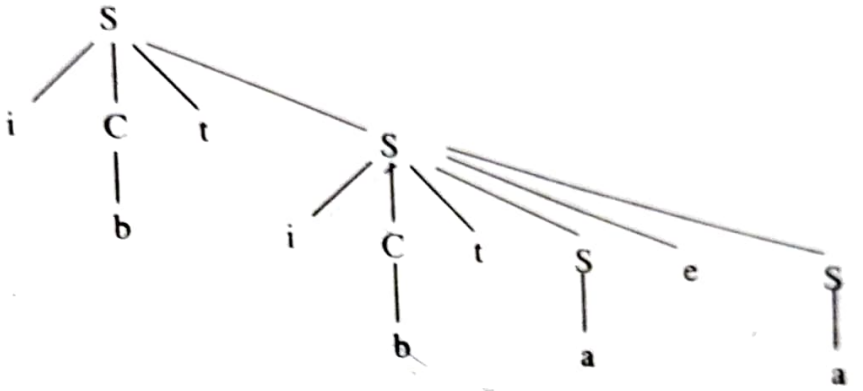
Example 5.19: Is the following grammar ambiguous?

$$
\begin{aligned}
S &\rightarrow iCtS \mid iCtSeS \mid a\\
C &\rightarrow b
\end{aligned}
$$

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

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

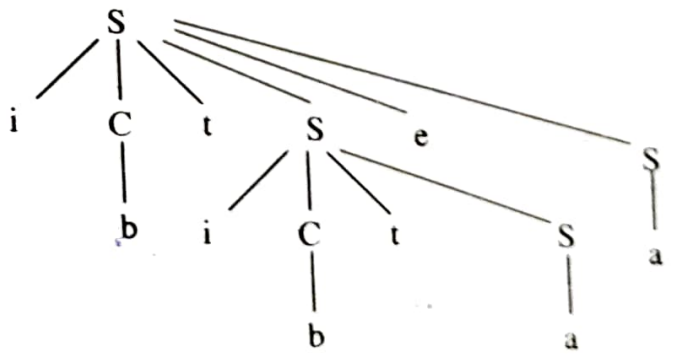
The parse tree for this is shown below:



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

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

The parse tree for this is shown below:



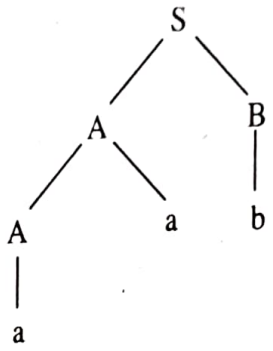
Since there are two different parse trees for the string 'ibtibtaea' by applying leftmost derivation the given grammar is ambiguous.

Example 5.20: Is the grammar ambiguous?

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

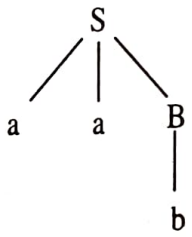
Consider the left most derivation for the string *aab* and the corresponding pares tree

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



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

$$S \Rightarrow aaB \Rightarrow aab$$



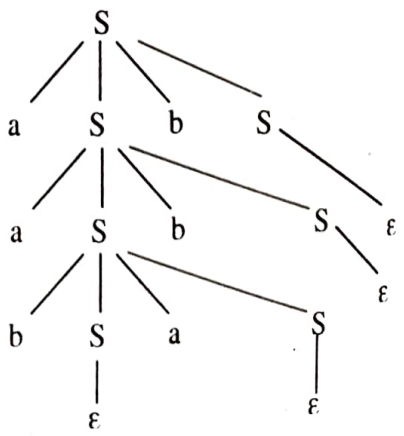
Since there are two parse trees for the string *aab*, the given grammar is ambiguous.

Example 5.21: Show that the following grammar is ambiguous

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

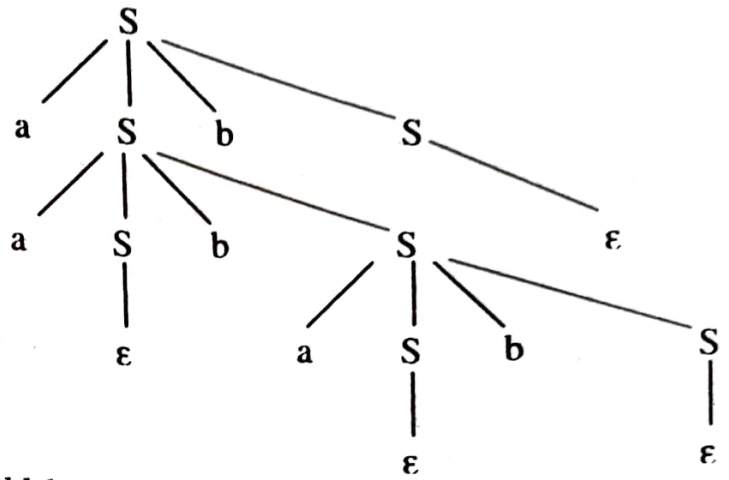
Consider the leftmost derivation for the string *aababb* and the corresponding parse tree

$$\begin{aligned}
S &\Rightarrow aSbS && \text{by using } S \rightarrow aSbS \\
&\Rightarrow aaSbSbS && \text{by using } S \rightarrow aSbS \\
&\Rightarrow aabSaSbSbS && \text{by using } S \rightarrow bSaS \\
&\Rightarrow aabaSbSbS && \text{by using } S \rightarrow \varepsilon \\
&\Rightarrow aababSbS && \text{by using } S \rightarrow \varepsilon \\
&\Rightarrow aababbS && \text{by using } S \rightarrow \varepsilon \\
&\Rightarrow aababb && \text{by using } S \rightarrow \varepsilon
\end{aligned}$$



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.

Eliminate the useless symbols in the grammar

1] $S \to aA \mid bB$
$A \to aA \mid a$
$B \to bB$
$D \to ab \mid Ea$
$E \to aC \mid d$

Soln Terminal symbols, $T = \{a, b, d\}$

$W_1 = \{A, E\}$

$W_2 = \{A, E, S, D\}$

$W_3 = \{A, E, S, D\}$

$G' = \{\{A, E, S, D\}, \{a, b, d\}, P, S\}$

$P: \{ S \to aA,$
$A \to aA \mid a$
$D \to ab \mid Ea$
$E \to d \}$

$Y_1 = \{S\}$

$Y_2 = \{S, A\}$

$Y_3 = \{S, A, a\}$

$Y_4 = \{S, A, a\}$

$G'' = \{\{S, A\}, \{a\}, P, S\}$

$P: \{ S \to aA,$
$A \to aA \mid a \}$

2) $S \to aA \mid a \mid Bb \mid cC$

$A \to aB$

$B \to a \mid Aa$

$C \to cCD$

$D \to ddd$

Sol 2 :- $T = \{a, c, d, b\}$

$W_1 = \{B, D\}$

$W_2 = \{B, D, A, S\}$

$W_3 = \{B, D, A, S\}$

$G' = \{\{B, D, A, S\}, \{a, b, c, d\}, P, S\}$

$P: \{ S \to aA \mid a \mid Bb,$

$A \to aB,$

$B \to a \mid Aa,$

$D \to ddd \}$

$y_1 = \{S\}$

$y_2 = \{S, A, B\}$

$y_3 = \{S, A, B, a\}$

$y_3 = \{S, A, B, a, b\}$

$G'' = \{\{S, A, B\}, \{a, b\}, P, S\}$

$P: \{ S \to aA \mid a \mid Bb$

$A \to aB,$

$B \to a \mid Aa \}$

3] $S \to AC \mid B$

$A \to a$

$C \to c \mid BC$

$E \to aA \mid e$

Sol:

Phase 1: Terminals, $T = \{a, c, e\}$

$W_1 = \{A, C, E\}$

$W_2 = \{A, C, E, S\}$

$W_3 = \{A, C, E, S\}$

$G_1' = \{\{A, C, E, S\}, \{a, c, e\}, P, S\}$

$P: \{ S \to AC,$

$A \to a,$

$C \to c,$

$E \to aA \mid e \}$

Phase 2:

$Y_1 = \{S\}$

$Y_2 = \{S, A, C\}$

$Y_3 = \{S, A, C, a, C\}$

$Y_4 = \{S, A, C, a, C\}$

$G_1'' = \{\{A, C, S\}, \{a, c\}, P, S\}$

$P: \{ S \to AC,$

$A \to a,$

$C \to c \}$

4) $T \to aaB \mid abA \mid aaT$

$A \to aA$

$B \to ab \mid b$

$C \to ad$

Phase 1 : Terminals, $T = \{a, b, d\}$

$W_1 = \{B, C\}$

$W_2 = \{B, C, T\}$

$W_3 = \{B, C, T\}$

$e_1' = \{\{B, C, T\}, \{a, b, d\}, P, S\}$

$P: \{ T \to aaB \mid aaT,$

$B \to ab \mid b,$

$C \to ad \}$

Phase 2 : $Y_1 = \{T\}$

$Y_2 = \{T, B\}$

$Y_3 = \{T, B, a, b\}$

$Y_4 = \{T, B, a, b\}$

$e_1'' = \{\{T, B\}, \{a, b\}, P, S\}$

$P: \{ T \to aaB \mid aaT,$

$B \to ab \mid b \}$

5] $S \to AB \mid a$
$A \to BC \mid b$
$B \to aB \mid C$
$C \to aC \mid B$

Soln :- Phase - 1:

Terminals, $T = \{a, b\}$

$W_1 = \{S, A\}$

$W_2 = \{S, A\}$

$e_1' = \{\{S, A\}, \{a, b\}, P, S\}$

$P: \{ S \to a,$
$A \to b \}$

Phase - 2 : $y_1 = \{S\}$

$y_2 = \{S, A\}$

$y_3 = \{S, A, a, b\}$

$\therefore y_4 = \{S, A, a, b\}$

$P: \{S \to a,$
$A \to b\}$ → A is not reachable

$P: \{S \to a\}$

Remove null productions

1) $S' \to S\#$
$S \to ABC$
$A \to a \mid bbD$
$B \to a \mid \epsilon$ ✓
$C \to b \mid \epsilon$ ✓
$D \to c \mid d$

first eliminate, $B \to \epsilon$

$S' \to S\#$
$S \to ABC \mid AC$
$A \to a \mid bbD$
$B \to a$
$C \to b \mid \epsilon$
$D \to c \mid d$

next eliminate, $C \to \epsilon$

$S' \to S\#$
$S \to ABC \mid AC \mid AB \mid A$
$A \to a \mid bbD$
$B \to a$
$C \to b$
$D \to c \mid d$

The grammar G',
$G' = (V', T', P', S)$
$V' = \{S, A, B, C, D, S'\}$
$T' = \{a, b, c, d\}$
$P : \{ S' \to S\#,$
$S \to ABC \mid AC \mid AB \mid A$
$A \to a \mid bbD$
$B \to a$
$C \to b$
$D \to c \mid d \}$
$S \to S'$ — start symbol

2] $S \to ABC \mid CbB \mid Ba$

$A \to da \mid BC$

$B \to gC \mid \epsilon$

$C \to ha \mid \epsilon$

first remove, $B \to \epsilon$

$S \to ABC \mid CbB \mid Ba \mid AC \mid Cb \mid a$

$A \to da \mid BC \mid C$

$B \to gC$

$C \to ha \mid \epsilon$

Remove, $C \to \epsilon$

$S \to ABC \mid CbB \mid Ba \mid AC \mid Cb \mid a \mid AB \mid bB \mid A \mid b$

$A \to da \mid BC \mid C \mid B$

$B \to gC \mid g$

$C \to ha$

3] $S \to ABCa \mid bD$

$A \to BC \mid b$

$B \to b \mid \epsilon$

$C \to c \mid \epsilon$

$D \to d$

first remove, $B \to \epsilon$

$S \to ABCa \mid bD \mid ACa$

$A \to BC \mid b \mid C$

$B \to b$

$C \to c \mid \epsilon$

$D \to d$

Eliminate, $C \to \epsilon$

$S \to ABCa \mid bD \mid ACa \mid ABa \mid Aa$

$B \to BC \mid b \mid C \mid B$

$B \to b$

$C \to c$

$D \to d$

4] $S \to BAAB$

$A \to 0A2 \mid 2A0 \mid \epsilon$

$B \to AB \mid 1B \mid \epsilon$

first remove, $A \to \epsilon$

$S \to BAAB \mid BAB \mid ABB \mid BA$

$A \to 0A2 \mid 2A0 \mid 02 \mid 20$

$B \to AB \mid 1B \mid B \mid \epsilon$

Eliminate $B \to \epsilon$

$S \to BAAB \mid BAB \mid AAB \mid BAA \mid AB \mid BB \mid BA \mid AA \mid A \mid B$

$A \to 0A2 \mid 2A0 \mid 02 \mid 20$

$B \to AB \mid 1B \mid B \mid A \mid 1$

$B \to B$ can be deleted.

$B \to AB \mid 1B \mid A \mid 1$

$G' = (V', T', P', S)$

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

$T' = \{0, 1, 2\}$

$P' = \{ S \to BAAB \mid BAB \mid AAB \mid BAA \mid AB \mid BB \mid BA \mid AA \mid A \mid B$

$A \to 0A2 \mid 2A0 \mid 02 \mid 20$

$B \to AB \mid 1B \mid A \mid 1$

$\}$

$S \to S.$

Remove unit productions

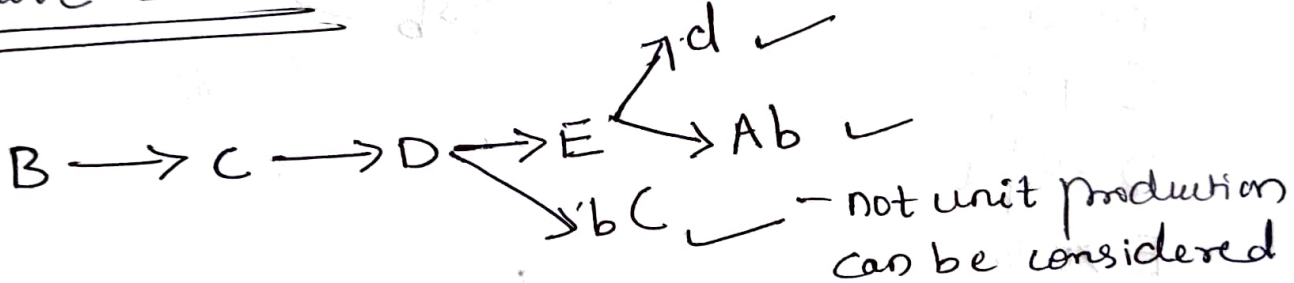
1)

$S \to AB$
$A \to a$
$B \to C \mid b$
$C \to D$
$D \to E \mid bC$
$E \to d \mid Ab$

unit productions,

$B \to C$
$C \to D$
$D \to E$

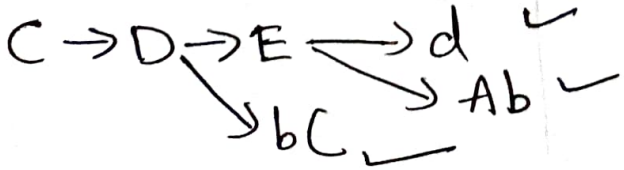
Remove $B \to C$



$S \to AB$
$A \to a$
$B \to d \mid Ab \mid bC \mid b$

$C \to D$
$D \to E \mid bC$
$E \to d \mid Ab$

Remove, $C \to D$



$S \to AB$

$A \to a$

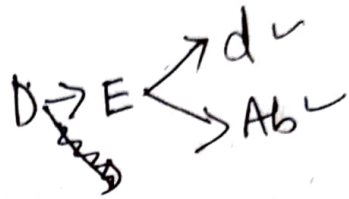
$B \to d \mid Ab \mid bC \mid b$

$C \to d \mid Ab \mid bC$

$D \to E \mid bC$

$E \to d \mid Ab$

Remove, $D \to E$



$S \to AB$

$A \to a$

$B \to d \mid Ab \mid bC \mid b$

$C \to d \mid Ab \mid bC$

$D \to d \mid Ab \mid bC$

$E \to d \mid Ab$

$G' = \{V', T', P', S\}$

$V' = \{S, A, B, C, D, E\}$

$T' = \{a, b, d\}$

$P' = \{S \to AB,$

$A \to a,$

$B \to d \mid Ab \mid bC \mid b,$

$C \to d \mid Ab \mid bC,$

$D \to d \mid Ab \mid bC,$

$E \to d \mid Ab\}$

$S = S \longrightarrow$ start Symbol

2] $S \to AO \mid B$

$B \to A \mid 11$

$A \to 0 \mid 12 \mid B$

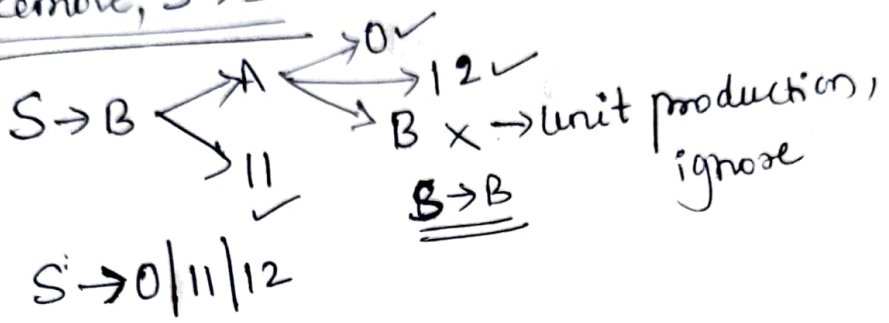
Unit productions,

$S \to B$

$B \to A$

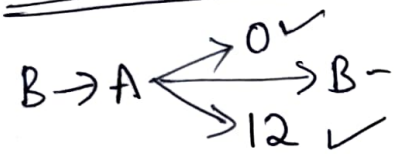
$A \to B$

Remove, $S \to B$



$S \to 0 \mid 11 \mid 12$

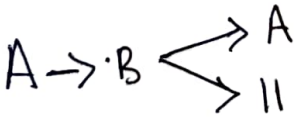
Remove; $B \to A$



$S \to AO \mid 0 \mid 11 \mid 12$

$B \to 0 \mid 12 \mid 11$

Remove, $A \to B$



$S \to AO \mid 0 \mid 11 \mid 12$

$B \to 0 \mid 12 \mid 11$

$A \to 0 \mid 12 \mid 11$

$G' = (V', T', P', S)$

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

$T' = \{0, 1, 2\}$

$P' = \{ S \to AO \mid 0 \mid 11 \mid 12,$

$B \to 0 \mid 12 \mid 11,$

$A \to 0 \mid 12 \mid 11 \}$

3] $S \to Aa \mid B \mid Ca$

$B \to aB \mid b$

$C \to Db \mid D$

$D \to E \mid d$

$E \to ab$

Unit productions,

$S \to B$

$C \to D$

$D \to E$

Remove, $S \to B$

$S \to Aa \mid Ca \mid aB \mid b$

$B \to aB \mid b$

$C \to Db \mid D$

$D \to E \mid d$

$E \to ab$

$S \to B \to aB$ ✓

$S \to B \to b$ ✓

Remove, $C \to D$

$C \to D \to E \to ab$ ✓

$C \to D \to d$ ✓

$S \to Aa \mid Ca \mid aB \mid b$

$B \to aB \mid b$

$C \to Db \mid ab \mid d$

$D \to E \mid d$

$E \to ab$

Remove, $D \to E$

$D \to E \to ab$

$S \to Aa \mid Ca \mid aB \mid b$

$B \to aB \mid b$

$C \to Db \mid ab \mid d$

$D \to ab \mid d$

$E \to ab$

$$e_1' = \{ V', T', P', S \}$$

$V' = \{ S, A, B, C, D, E \}$

$T' = \{ a, b, d \}$

$P' = \{ S \to Aa \mid Ca \mid aB \mid db,$

$B \to aB \mid b,$

$C \to Db \mid ab \mid d,$

$D \to d \mid ab,$

$E \to ab \}$

$S \to S$, Start symbol

Chomsky Normal Form (CNF)

$A \to BC$

no. of terminals on right hand side is restricted to 2. (not more than 2).

or

$A \to a$

$a \in$ terminal

$A, B, C \in$ non-terminals

(1) CFG to CNF

$S \to ASA \mid aB$

$A \to B \mid S$

$B \to b \mid \epsilon$

(i) Start symbol occurs on right side, so we will consider new symbol as S'.

$S' \to S$

(ii) Remove null productions

$B \to \epsilon$, remove it

$S' \to S$
$S \to ASA \mid aB \mid a$
$A \to B \mid S \mid \epsilon$ ← null production
$B \to b$

$A \to \epsilon$

$S' \to S$
$S \to ASA \mid aB \mid a \mid SA \mid AS \mid S$
$A \to B \mid S$
$B \to b$

(iii) Remove unit productions

$A \to B$
$A \to S$
$S' \to S$
$S \to S$ → will remove directly

$S' \to S$
$S \to ASA \mid aB \mid a \mid SA \mid AS$
$A \to B \mid S$
$B \to b$

$S' \to S$

$S' \to ASA \mid aB \mid a \mid SA \mid AS$
$S \to ASA \mid aB \mid a \mid SA \mid AS$
$A \to B \mid S$
$B \to b$

$A \to B$,

$S' \to ASA \mid aB \mid a \mid SA \mid AS$

$S \to ASA \mid aB \mid a \mid SA \mid AS$

$A \to b \mid S$ $\qquad$ $A \to B \to b$

$B \to b$

$A \to S$,

$S' \to ASA \mid aB \mid a \mid SA \mid AS$

$S \to ASA \mid aB \mid a \mid SA \mid AS$

$A \to b \mid ASA \mid aB \mid a \mid SA \mid AS$

$B \to b$

$A \to S$: ASA, aB, a, SA, AS

ASA → not in CNF form (should contain only 2 nonterminals on right side)

(iv) ASA

Replacing, SA by X,

$X \to SA$

$S' \to AX \mid aB \mid a \mid SA \mid AS$

$S \to AX \mid aB \mid a \mid SA \mid AS$

$A \to b \mid AX \mid aB \mid a \mid SA \mid AS$

$B \to b$

aB → not in CNF form,

Replace terminal symbol by non-terminal symbol.

$S' \to AX \mid aB \mid a \mid SA \mid AS$

$aB \to Y$

$Y \to a$

$S' \to AX \mid YB \mid a \mid AS \mid SA$

$S \to AX \mid YB \mid a \mid AS \mid SA$

$A \to b \mid AX$

$B \to b$

$Y \to a$

$X \to SA$

2] $S \to a \mid aA \mid B$

$A \to aBB \mid \epsilon$

$B \to Aa \mid b$

Remove null productions, $A \to \epsilon$

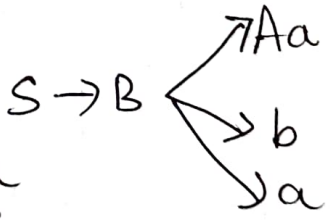
$S \to a \mid aA \mid B$

$A \to aBB$

$B \to Aa \mid b \mid a$

Remove unit productions

$S \to B$



$S \to a \mid aA \mid Aa \mid b \mid a$

$A \to aBB$

$B \to Aa \mid b \mid a$ not in CNF form

Replace, terminal with non-terminal,

$X \to a$

$S \to a \mid XA \mid AX \mid b \mid a$

$A \to XBB$ not in CNF.

$B \to AX \mid b \mid a$

Replace BB with Y,

Y → BB.

S → a | XA | AX | b | a

S → XY

B → AX | b | a now, it is in CNF

3] S → ABa

A → aab

B → Ac not in CNF

Replace, terminal with nonterminal.

X → a, Y → b, Z → c

S → ABX

A → XXY → not in CNF

B → AZ

Replace, AB with T, XX with V.

T → AB

V → XX

S → TX

A → VY

B → AZ

Now, in CNF

4]

$S \to 0A \mid 1B$

$A \to 0AA \mid 1S \mid 1$

$B \to 1BB \mid 0S \mid 0$

$S \to 0A \mid 1B$

$A \to 0AA \mid 1S \mid 1$

$B \to 1BB \mid 0S \mid 0$ not in CNF.

Replace, terminal with non-terminal,

$X \to 0$

$Y \to 1$

$S \to XA \mid YB$

$A \to XAA \mid YS \mid 1$

$B \to YBB \mid XS \mid 0$ not in CNF.

$S \to XA \mid YB$

$A \to XZ \mid YS \mid 1$

$B \to YV \mid XS \mid 0$

$X \to 0$

$Y \to 1$

$Z \to AA$

$V \to BB$.

Replace,

$Z \to AA$

$V \to BB$

5) $S \to bA \mid aB$

$A \to bAA \mid as \mid a$

$B \to aBB \mid bs \mid b$

(i) S is appearing on right hand side.
↓ start symbol. Introduce a new symbol S'

$S' \to S$

(ii) Remove null productions

none of the production contains null productions

(iii) Remove unit productions

~~none of these productions contains unit productions~~.

(iv) $S' \to S$

$S' \to S \to bA$, $S' \to S \to aB$

$S' \to bA \mid aB$

$A \to bAA \mid as \mid a$

$S \to bA \mid aB$

$B \to aBB \mid bS \mid b$ not in CNF.

(iv) Replace terminal with non-terminal

$X \to a$, $Y \to b$

$S' \to YA \mid XB$

$A \to YAA \mid XS \mid a$

$S \to YA \mid XB$

$B \to XBB \mid YS \mid b$ not in CNF

$X \to a$

$Y \to b$

(v) Replace AA with T, BB with R

$T \to AA$, $R \to BB$

$S' \to YA \mid XB$

$A \to YT \mid XS \mid a$

$S \to YA \mid XB$

$B \to XR \mid YS \mid b$

$X \to a$

$Y \to b$

6] $S \to ABA$

$A \to aA \mid \epsilon$

$B \to bB \mid \epsilon$

(i). Start symbol, S is not appearing on right hand side, so, no need to introduce new symbol

(ii) Remove null productions.

$A \to \epsilon$

$S \to ABA \mid BA \mid AB \mid B$

$A \to aA \mid a$

$B \to bB \mid \epsilon$

$B \to \epsilon$

$S \to ABA \mid BA \mid AB \mid B \mid AA \mid A$

$A \to aA \mid a$

$B \to bB \mid b$

(iii) Remove unit productions

$S \to B$

$S \to A$

$S \to B$,

$S \to B \to bB$

$S \to B \to b$

$S \to ABA \mid BA \mid AB \mid bB \mid AA \mid A \mid b$

$A \to aA \mid a$

$B \to bB \mid b$

$S \to A$,

$S \to A \to aA$

$S \to A \to a$

$S \to ABA \mid BA \mid AB \mid bB \mid AA \mid aA \mid a \mid b$

$A \to aA \mid a$

$B \to bB \mid b$

not in CNF

(iv) Replace terminals with non-terminals.

$X \to a$,

$Y \to b$

$S \to ABA \mid BA \mid AB \mid YB \mid AA \mid XA \mid a \mid b$

$A \to XA \mid a$

not in CNF

$B \to YB \mid b$

$X \to a$

$Y \to b$

(V) Replace, AB with Z.

$S \rightarrow ZA \mid BA \mid AB \mid YB \mid AA \mid XA \mid a \mid b$

$A \rightarrow XA \mid a$

$B \rightarrow YB \mid b$

$X \rightarrow a$

$Y \rightarrow b$

$Z \rightarrow AB$

Example 7.59: Convert the following grammar

$$A \rightarrow BC$$
$$B \rightarrow CA \mid b$$
$$C \rightarrow AB \mid a$$

into GNF

Solution:

Let $A = A_1$, $B = A_2$, $C = A_3$ and the resulting grammar is

$$A_1 \rightarrow A_2A_3$$
$$A_2 \rightarrow A_3A_1 \mid b$$
$$A_3 \rightarrow A_1A_2 \mid a$$

First two productions are of the form: $A_i \rightarrow A_j\alpha$ where $i < j$. So, we consider A_3-production.

Consider A_3-production: Substituting for A_1 in A_3-production we get

$$A_3 \rightarrow A_1A_2 \mid a = (A_2A_3)A_2 \mid a = A_2A_3A_2 \mid a$$

Again replacing the first A_2 in A_3-production we get,

$$A_3 \rightarrow A_2A_3A_2 \mid a = (A_3A_1 \mid b)\, A_3A_2 \mid a$$
$$= A_3A_1A_3A_2 \mid bA_3A_2 \mid a$$

we get the resulting A_3-production as

$$A_3 \rightarrow A_3A_1A_3A_2 \mid bA_3A_2 \mid a$$

which is having left recursion. After eliminating left recursion, we get

$$A_3 \rightarrow bA_3A_2 \mid a \mid bA_3A_2Z \mid aZ$$
$$Z \rightarrow A_1A_3A_2 \mid A_1A_3A_2Z$$

Now, all A_3-productions are in GNF.

Consider A_2-productions: Since all A_3-productions are in GNF, substituting A_3-productions in A_2-production we get

$$A_2 \rightarrow (bA_3A_2 \mid a \mid bA_3A_2Z \mid aZ)\, A_1 \mid b$$
$$= bA_3A_2A_1 \mid aA_1 \mid bA_3A_2ZA_1 \mid aZA_1 \mid b$$

which is in GNF. Now, all A_2-productions are in GNF.

Consider Z-productions: Since A_1-productions are in GNF, substituting A_1-production in Z-production we get Z-productions also in GNF as shown below:

$$Z \rightarrow A_1A_3A_2 \mid A_1A_3A_2Z$$
$$= (bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3A_2ZA_1A_3 \mid aZA_1A_3 \mid bA_3)\, A_3A_2$$
$$(bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3A_2ZA_1A_3 \mid aZA_1A_3 \mid bA_3)\, A_3A_2Z$$

which can be written as

$$Z \rightarrow bA_3A_2A_1A_3A_3A_2 \mid aA_1A_3A_3A_2 \mid bA_3A_2ZA_1A_3A_3A_2 \mid$$
$$aZA_1A_3\ A_3A_2 \mid bA_1A_3A_2$$
$$Z \rightarrow bA_3A_2A_1A_3A_3A_2\ Z \mid aA_1A_3A_3A_2\ Z \mid bA_3A_2ZA_1A_3A_3A_2Z \mid$$
$$aZA_1A_3\ A_3A_2Z \mid bA_3A_3A_2Z$$

Since all productions are in GNF, the resulting grammar is also in GNF. So, the final grammar obtained in GNF notation is

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

where

$$V = \{A_1, A_2, A_3, Z\}$$
$$T = \{a, b\}$$

$$P = \{$$
$$A_1 \rightarrow bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3A_2ZA_1A_3 \mid aZA_1A_3 \mid bA_3$$
$$A_2 \rightarrow bA_3A_2A_1 \mid aA_1 \mid bA_3A_2ZA_1 \mid aZA_1 \mid b$$
$$A_3 \rightarrow bA_3A_2 \mid a \mid bA_3A_2Z \mid aZ$$
$$Z \rightarrow bA_3A_2A_1A_3A_3A_2 \mid aA_1A_3A_3A_2 \mid bA_3A_2ZA_1A_3A_3A_2 \mid$$
$$aZA_1A_3\ A_3A_2 \mid bA_3A_3A_2$$
$$Z \rightarrow bA_3A_2A_1A_3A_3A_2Z \mid aA_1A_3A_3A_2\ Z \mid bA_3A_2ZA_1A_3A_3A_2Z \mid$$
$$aZA_1A_3\ A_3A_2Z \mid bA_3A_3A_2Z$$
$$\}$$

A_1 is the start symbol

Example 7.60: Convert the following grammar

$$S \rightarrow AA \mid 0$$
$$A \rightarrow SS \mid 1$$

into GNF

Solution:

Let $S = A_1$ and $A = A_2$. After substitution, the resulting grammar obtained is shown below:

$$A_1 \rightarrow A_2A_2 \mid 0$$
$$A_2 \rightarrow A_1A_1 \mid 0$$

First production is of the form: $A_i \rightarrow A_j\alpha$ where $i < j$. So, we consider A_2-production.

Consider A_2-production: Substituting for A_1 in A_2-production we get

$$A_2 \rightarrow A_2A_2\ A_1 \mid 0A_1 \mid 1$$

The above grammar is having left recursion. After eliminating left recursion, we get

$$A_2 \rightarrow 0A_1 \mid 1 \mid 0A_1Z \mid 1Z$$

$$Z \rightarrow A_2A_1 \mid A_2A_1Z$$

Now, all A_2-productions are in GNF.

Consider A_1-productions: Since all A_2-productions are in GNF, substituting A_2-productions in A_1-production we get

$$A_1 \rightarrow 0A_1A_2 \mid 1A_2 \mid 0A_1Z\,A_2 \mid 1Z\,A_2 \mid 0$$

which is in GNF. Now, all A_1-productions are in GNF.

Consider Z-productions: Since A_2-productions are in GNF, substituting A_2-production in Z-production we get Z-productions also in GNF as shown below:

$$Z \rightarrow 0A_1A_1 \mid 1A_1 \mid 0A_1Z\,A_1 \mid 1ZA_1$$

$$Z \rightarrow 0A_1A_1Z \mid 1A_1Z \mid 0A_1ZA_1Z \mid 1ZA_1Z$$

So, the final grammar obtained which is in GNF is shown below:

$$A_1 \rightarrow 0A_1A_2 \mid 1A_2 \mid 0A_1Z\,A_2 \mid 1Z\,A_2 \mid 0$$

$$A_2 \rightarrow 0A_1 \mid 1 \mid 0A_1Z \mid 1Z$$

$$Z \rightarrow 0A_1A_1 \mid 1A_1 \mid 0A_1Z\,A_1 \mid 1ZA_1$$

$$Z \rightarrow 0A_1A_1Z \mid 1A_1Z \mid 0A_1ZA_1Z \mid 1ZA_1Z$$

Exercises

1) Define the following: i) Context-free grammar ii) Derivation

2) What is a sentence or sentential form? Explain with example.

3) Consider the following grammar

$$S \rightarrow aCa$$

$$C \rightarrow aCa \mid b$$

What is the language generated by this grammar?

4) Obtain a grammar to generate the following languages:

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

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

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

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

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

5) Obtain a grammar to generate set of all palindromes over $\Sigma = \{a, b\}$.

6) Obtain a grammar to generate $L = \{ww^R \text{ where } w \in \{a, b\}^*\}$.

7) Obtain a grammar to generate strings of all non-palindromes over $\{a, b\}$.

8) Obtain the grammar to generate following languages:

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

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

9) What is the language generated by the following grammar

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

$A \rightarrow 1S$

10) Obtain a CFG to generate a string of balanced parentheses.

11) Obtain a grammar to generate the following languages:

a) $L = \{0^i 1^j \mid i \neq j, i \geq 0 \text{ and } j \geq 0\}$

b) $L = \{a^{n+2} b^m \mid n \geq 0 \text{ and } m > n\}$

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

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

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

f) $L = L_1 \cup L_2$ where $L_1 = \{a^n b^m \mid n \geq 0, m > n\}$ $L_2 = \{0^n 1^{2n} \mid n \geq 0 \}$

g) $L = \{w : |w| \bmod 3 \neq |w| \bmod 2\}$ on $\Sigma = \{a\}$

h) $L = \{w : |w| \bmod 3 \geq |w| \bmod 2\}$ on $\Sigma = \{a\}$

12) Obtain a grammar to generate set of all strings with exactly one *a* when $\Sigma = \{a, b\}$.

13) Obtain a grammar generating all strings with at least one *a* if $\Sigma = \{a, b\}$.

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

15) Obtain a grammar to generate the following languages:

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

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

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

d) Strings of 0's and 1's having a substring 000

e) $L = \{a^{n1} b^{n1} a^{n2} b^{n2} \ldots\ldots a^{nk} b^{nk} : n, k \geq 0\}$

16) Obtain a grammar to generate integers. Show the derivation for the unsigned number 1965 and signed number +1965.

17) Is the following grammar ambiguous?

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

18) Show that the following grammar is ambiguous.

$S \rightarrow SbS \mid a$

19) Given the following grammar:

$E \rightarrow I \mid E+E \mid E*E \mid (E)$

$I \rightarrow a \mid b \mid c \mid d$

Obtain the derivations for the strings (a+b)*c*d and a+b*c and the parse tree for each derivation.

20. Obtain a reduced grammar for the grammar shown below:

$S \rightarrow aAa$

$A \rightarrow Sb \mid bCC \mid aDA$

$C \rightarrow ab \mid aD \mid$

$E \rightarrow aC$

$D \rightarrow aAD$

21. Find an equivalent grammar without ε-productions for the grammar shown below.

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

$A \rightarrow aBb \mid bBa$

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

22. Remove the unit productions from the grammar

$S \rightarrow A \mid B \mid Cc$

$A \rightarrow aBb \mid B$

$B \rightarrow aB \mid bb$

$C \rightarrow Cc \mid B$

23. Eliminate useless productions from the grammar

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

$A \rightarrow aB \mid \varepsilon$

$B \rightarrow aA$

$C \rightarrow CcD$

$D \rightarrow abd$

Note: First remove ε and unit productions, then simplify CFG.

24. Obtain the following grammar in CNF

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid I$

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

25. Obtain the following grammar in CNF and GNF

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

$A \rightarrow aB \mid \varepsilon$

$B \rightarrow aA|$

$C \rightarrow cCD$

$D \rightarrow abd$

26. Simplify the following CFG and convert it into CNF

$S \rightarrow AaB \mid aaB$

$A \rightarrow \varepsilon$

$B \rightarrow bbA \mid \varepsilon$

27. Convert the following grammar into GNF

$S \rightarrow abAB$

$A \rightarrow bAB \mid \varepsilon$

$B \rightarrow Aa \mid \varepsilon$

28. What is left recursion? How it can be eliminated?

29. Eliminate left recursion from the following grammar

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid id$

30. Eliminate the useless symbols in the grammar

$S \rightarrow aA \mid bB$

$A \rightarrow aA \mid a$

$B \rightarrow bB$

$D \rightarrow ab \mid Ea$

$E \rightarrow aC \mid d$

31. Eliminate left recursion from the following grammar

$S \rightarrow Ab \mid a$

$A \rightarrow Ab \mid Sa$

32. What is the need for simplifying a grammar?

33. What is a useless symbol? Explain with example.

34. Simplify the following grammar

$S \rightarrow aA \mid a \mid Bb \mid cC$

$A \rightarrow aB$

$B \rightarrow a \mid Aa$

$C \rightarrow cCD$

$D \rightarrow ddd$

35. What is an ε - production? What is a Nullable variable? Explain with example.

36. Eliminate all ε- productions from the grammar

$S \rightarrow ABCa \mid bD$

$A \rightarrow BC \mid b$

$B \rightarrow b \mid \varepsilon$

$C \rightarrow c \mid \varepsilon$

$D \rightarrow d$

37. Eliminate all ε- productions from the grammar

$S \rightarrow BAAB$

$A \rightarrow 0A2 \mid 2A0 \mid \varepsilon$

$B \rightarrow AB \mid 1B \mid \varepsilon$

38. 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}

39. Define the following terms:

a) Derivation tree b) yield of a tree c) ambiguous grammar

40. Is the following grammar ambiguous?

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

$C \rightarrow b$

41. What is dangling else problem? How dangling else problem can be solved?"

42. Eliminate ambiguity from the following ambiguous grammar:

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

$C \rightarrow b$

43. Convert the following ambiguous grammar into unambiguous grammar

$E \rightarrow E * E \mid E - E$

$E \rightarrow E \wedge E \mid E / E$

$E \rightarrow E + E$

$E \rightarrow (E) \mid \mathbf{id}$

44. What is inherently ambiguous grammar?"

45. Obtain the inherently ambiguous grammar for the following inherently ambiguous languages:

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

$L = \{a^i b^j c^k \mid i, j, k, \geq 0\}$ and $i = j$ or $j = k\}$

46. What is left recursion? How to eliminate left recursion?

47. What is useless variable? Explain with example.