UNIT-IV

Regular Grammars:

Grammar:

- A grammar is defined as **G = (V, Σ, P, S)**. Where
  - ✓ **V** is a finite nonempty set whose elements are called ***variables***.
  - ✓ **Σ** is a finite nonempty set whose elements are called ***Terminals***.
  - ✓ **P** is a finite set whose elements are α→β, where α and β are strings on **V** U Σ. And α has at least one symbol from **V**. Elements of **P** are called ***productions*** or ***production rules*** or ***rewriting rules***.
  - ✓ **S** is a special variable i.e. an element of **V** called the ***start symbol*** and
  - ✓ **V ∩ Σ = ϕ**
- We observe the following regarding the production rules:
  - ✓ Reverse substitutions is not permitted. For example if S → AB is a production then we can place S by AB but we cannot replace AB by S.
  - ✓ No inversion operation is permitted. For example if S → AB is a production then, it is not necessary that AB → S is a production.

Example:

- **G = (V, Σ, P, S) is a grammar**

  Where

  V = {<sentence>, <noun>, <verb>, <adverb>}

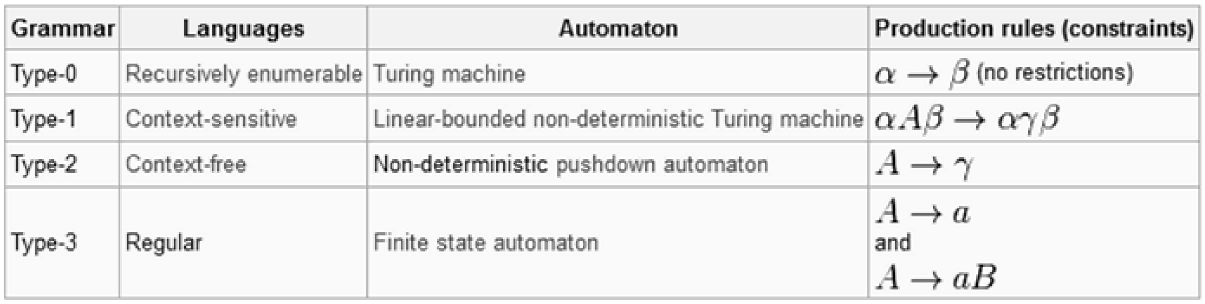
  Σ = {Ram, Sam, ate, sang, well}

  S = <sentence>
- Then **P** consists of the following productions:
  - ✓ <sentence> → <noun> <verb>
  - ✓ <sentence> → <noun> <verb> <adverb>
  - ✓ <noun> →Ram
  - ✓ <noun> → Sam
  - ✓ <verb> → ate
  - ✓ <verb> → sang
  - ✓ <adverb> → well

  <u>**Note**</u>: we use comma "**,**" as a separator to separate multiple productions and alternation "|" to put several productions together.

- **Derivation:** Derivation is an ordered tree which is defined as sequence of replacements of a substring in a sentential form. Productions are used to derive one string over V U Σ from another string.

- The formal definition of derivation is as follows:
  - ✓ If α→β is a production in a grammar G and ϒ, δ are any two strings on V U Σ, then we say ϒαδ directly derives ϒβδ in G. this process is called as one step derivation.
- For example
  - ✓ G = ({S}, {0, 1}, {S → 0S1, S → 01} has the productions S → 0S1. So ***S*** in 0***S***1 can be replaced by ***0S1*** i.e. now S → 0***0S1***1.
- In the definition of grammar **(V, Σ, P, S), V** and **Σ** are sets of symbols and S Є **V**. So if we want to classify grammars, we have to do it only by considering the form of productions.
- Chomsky classified the grammars into four types in terms of productions (types 0-3) which is as shown in the table below:



Grammar	Languages	Automaton	Production rules (constraints)
Type-0	Recursively enumerable	Turing machine	$\alpha \rightarrow \beta$ (no restrictions)
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine	$\alpha A\beta \rightarrow \alpha\gamma\beta$
Type-2	Context-free	Non-deterministic pushdown automaton	$A \rightarrow \gamma$
Type-3	Regular	Finite state automaton	$A \rightarrow a$ and $A \rightarrow aB$

- A **Type-0** grammar is any phrase structure grammar or simply a grammar without any restrictions. Type-0 grammars (unrestricted grammars) include all formal grammars. They generate exactly all languages that can be recognized by a Turing machine. These languages are also known as the recursively enumerable languages.
- A grammar is called **Type-1** grammar or Context sensitive grammar or context dependent grammar if all its productions are Type-1 productions. Type -1 production is of the form αAβ → αϒβ. Where A is a non-terminal or variable. α, β and ϒ are the strings of terminals and non-terminals.
- The symbols α and β may be empty but ϒ must be non empty. And also the production of the form S → ε is allowed if S does not appear on the right hand side of the production.
- A language generated by the Type-1 grammar is called as Type-1 or context sensitive language.
- A **Type-2** grammar or context free grammar is a grammar if it contains only Type-2 productions Type-2 production is of the form A → ϒ, where A Є V and ϒ is a string of terminals and non-terminals i.e. ϒ Є (V U Σ)*.
- A language generated by the Type-2 grammar is called as Type-2 or context free language.
- **Type-3** grammars (regular grammars) generate the regular languages. Such a grammar restricts its rule to a single non-terminal on the left-hand side and a right-hand side consisting of a single terminal, possibly followed or precedes, but not both in the same grammar by a single non-terminal. A production S → ε is allowed in Type-3 grammar, but in this case S does not appear on the right hand side of any production.

Regular Grammar

A regular grammar is a formal grammar that describes the regular language. Where a formal grammar is defined as a set of rules for rewriting the strings, along with a start symbol from which the rewriting must start.

Therefore, a grammar is usually thought of as a language generator. However, it can also sometimes be used as the basis for a "recognizer"—a function in computing that determines whether a given string belongs to the language or is grammatically incorrect.

To describe such recognizers, formal language theory uses separate formalisms, known as automata theory. One of the interesting results of automata theory is that it is not possible to design a recognizer for certain formal languages. Parsing is the process of recognizing an utterance i.e. expression or word (a string in natural languages) by breaking it down to a set of symbols and analyzing each one against the grammar of the language.

The regular grammars are of two types: 1) left regular grammars and 2) right regular grammars.

Right Regular Grammar

- Right regular grammar is also called as right linear grammar which is a formal grammar **(V, Σ, P, S)** such that all the productions or production rules in P are of one of the following forms:
  - ✓ B → a (Where B is a variable or non-terminal in V and *a* is terminal in Σ)
  - ✓ B → aC (Where B and C are in a Variable V and a is terminal in Σ)
  - ✓ B → ε (Where B is a variable in V and ε denotes the empty string i.e. the string of length 0).

Left Regular Grammar

- Left regular grammar is also called as left linear grammar which is a formal grammar **(V, Σ, P, S)** such that all the productions or production rules in P are of one of the following forms
  - ✓ A → *a* d(Where A is a non-terminal in V and *a* is a terminal in Σ)
  - ✓ A → B*a* (Where A and B are in V and *a* is in Σ)
  - ✓ A → ε (Where A is a variable in V and ε is the empty string)

Example:

- An example of a right regular grammar G with N = {S, A}, Σ = {a, b, c}, P consists of the following rules:

  S → *a*S

  S → bA

$A \rightarrow \varepsilon$

$A \rightarrow cA$

And S is the start symbol. This grammar describes the same language as the regular expression a*bc*

Extended Regular Grammars

- An extended right regular grammar is one in which all rules obey one of the following:
  - ✓ $B \rightarrow a$ (Where B is a non-terminal in V and *a* is a terminal in Σ)
  - ✓ $A \rightarrow wB$ (Where A and B are in V and w is in Σ*)
  - ✓ $A \rightarrow \varepsilon$ (Where A is in V and ε is the empty string)
- An extended right regular grammar is also called as strictly right regular grammar.
- An extended left regular grammar is one in which all rules obey one of the following:
  - ✓ $A \rightarrow a$ (Where A is a non-terminal in V and *a* is a terminal in Σ)
  - ✓ $A \rightarrow Bw$ (Where A and B are in V and w is in Σ*)
  - ✓ $A \rightarrow \varepsilon$ (Where A is in V and ε is the empty string)
- An extended left regular grammar is also called as strictly left regular grammar.

Examples:

Let us consider the grammar G=({S},{a, b},R,S), where R=

$S \rightarrow abS$

$S \rightarrow \lambda$

$S \rightarrow Sab$

EQUIVALENCE BETWEEN REGULAR LINEAR GRAMMAR AND FINITE AUTOMATA, INTERCONVERSION

- The equivalence exists between regular grammar and finite automata in accepting languages.

CONSTRUCTION OF REGULAR GRAMMAR GENERATING T(M) FOR A GIVEN DFA M

- Let M = ({q_0, … ,q_n}, Σ, δ, q_0,F). if w is a string in the language of machine M i.e. T(M), then it is obtained by concatenating the labels corresponding to several transitions, the first from q_0 and the last terminating at some final state.
- So for the grammar G to be constructed, productions should correspond to transitions. Also, there should be provision for terminating the derivation tree once a transition terminating at some final state is encountered.
- With these ideas we construct G as:

$$G = (\{A_0, A_1, \dots, A_n\}, \Sigma, P, A_0)$$

P is defined by the following rules:

(1) $A_i \rightarrow aA_i$ is included in P if $\delta(q_i, a) = q_j \notin F$

(2) $A_i \rightarrow aA_j$ and $A_i \rightarrow a$ are included in P if $\delta(q_i, a) = q_j \in F$.

We can now show that L(G) = T(M) by using the construction of P. Such a construction gives:

$$A_i \rightarrow aA_j \quad \text{iff } \delta(qi, a) = qj$$

$$A_i \rightarrow a \quad \text{iff } \delta(qi, a) = F$$

So $A_0 \rightarrow a_1A_1 \rightarrow a_1a_2A_2 \rightarrow \ldots \rightarrow a_1 \ldots \rightarrow a_{k-1}A_k \rightarrow a_1a_2 \ldots a_k$ iff

$\delta(q_0, a_1) = q_1, \quad \delta(q_1, a_2) = q_2, \ldots, \delta(q_k, a_k) \in F$

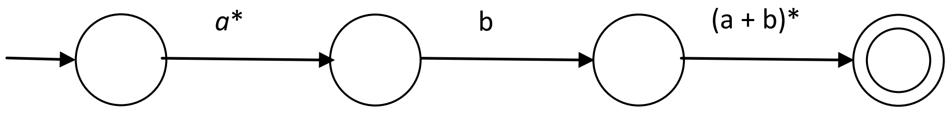
This proves that w = a1 . . . ak ∈ L(G) iff $\delta(q_0, a_1 \ldots a_k) \in F$, i.e. iff w ∈ T(M).

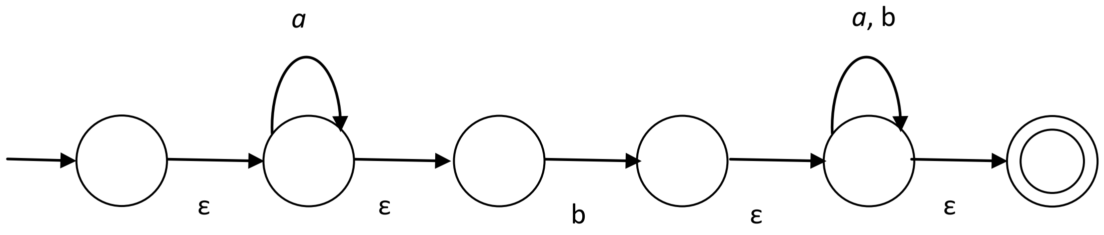
Example:

- Construct a regular grammar G generating the regular set represented by P = a*b(a + b)*.

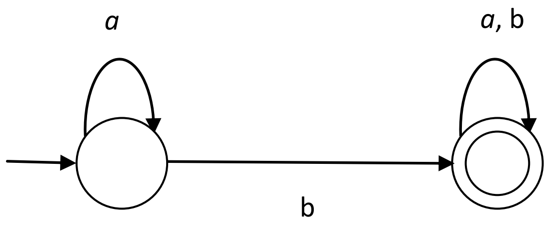
Solution:

- We construct the DFA corresponding to P using the subset method.





After eliminating ε – moves we get the DFA straightaway as shown in figure given below:



Let G = ({A_0, A_1}, {a, b}, P, A_0), where P is given by

$A_0 \rightarrow aA_0, \qquad A_0 \rightarrow bA_1, \qquad A_0 \rightarrow b$

$A_1 \rightarrow aA_1$, $A_1 \rightarrow bA_1$, $A_1 \rightarrow a$, $A_1 \rightarrow b$

➢ Hence G is the required regular grammar.

CONSTRUCTION OF TRANSITION SYSTEM M ACCEPTING L(G) FOR A GIVEN REGULAR GRAMMAR G

➢ Let us consider the grammar $G = (\{A_0, A_1, \dots, A_n\}, \Sigma, P, A_0)$. We construct the transition system M whose
(a) States corresponds to variables
(b) Initial state corresponds to A_0
(c) Transition in M corresponds to productions in P.

➢ As the last production applied in any derivation is of the form $A_i \rightarrow a$, the corresponding transition terminates at a new state and this is the unique final state.

Now

We define M as $(\{q_0, \dots, q_n\}, \Sigma, \delta, q_0, q_f)$. δ is defined as follows:
(1) Each production $A_i \rightarrow aA_j$ induces a transition from q_i to q_j with label a.
(2) Each production $A_k \rightarrow a$ induces a transition from q_k to q_f with label a.

From the construction it is easy to see that $A_0 \rightarrow a_1A_1 \rightarrow a_1a_2A_2 \rightarrow \dots \rightarrow a_1 \dots \rightarrow a_{n-1}A_{n-1} \rightarrow a_1 \dots a_n$ is a derivation iff there is a path in M starting from q_0 and terminating in q_f with path values $a_1a_2 \dots a_n$. Therefore L(G) = T(M)

Example:

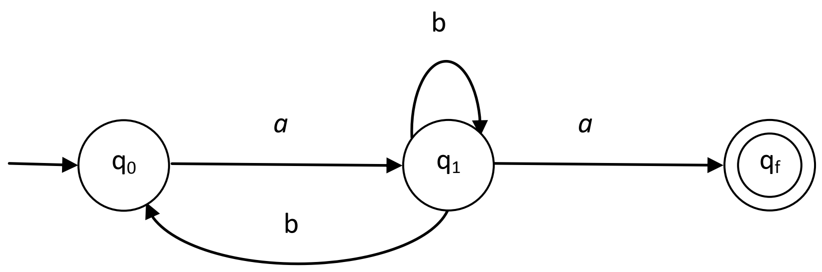
➢ Let $G = (\{A_0, A_1\}, \{a, b\}, P, A_0)$, where P consists of $A_0 \rightarrow aA_1$, $A_1 \rightarrow bA_1$, $A_1 \rightarrow a$, $A_1 \rightarrow bA_0$. Construct a transition system M accepting L(G)

Solution:

➢ Let $M = (\{q_0, q_1, q_f\}, \{a, b\}, \delta, q_0, \{q_f\})$, where q_0 and q_f correspond to A_0 and A_1, respectively and q_f is the new (final) state introduced.

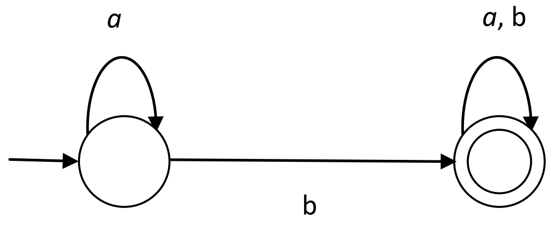
➢ $A_0 \rightarrow aA_1$ induces a transition from q_0 to q_1 with label a. Similarly, $A_1 \rightarrow bA_1$ and $A_1 \rightarrow bA_0$ induce transitions from q_1 to q_1 with label b and from q_1 to q_0 with label b, respectively.

➢ $A_1 \rightarrow a$ induces a transition from q_1 to q_f with label a. M is given a follows:



- Now Let G = ({A_0, A_1}, {*a*, b}, P, A_0), where P consists of $A_0 \to aA_0$, $A_0 \to bA_1$, $A_0 \to b$ $A_1 \to aA_1$, $A_1 \to bA_1$, $A_1 \to a$, $A_1 \to b$

Solution:



CONTEXT FREE GRAMMAR

- Context free grammars have played a central role in compiler technology since the 1960's; they turned implementation of parsers (functions that discover the structure of a program) from a time consuming, ad-hoc implementation task into a routine job.

An informal example

- Let us consider the language of palindromes. A palindrome is a string that reads the same forward and backward, such as **otto** or **madamimadam ("Madam, I'm Adam").**
- Put another way, string w is palindrome if and only if $w = w^R$ (reverse of string). To make things simple, we shall consider describing only the palindromes with alphabet {0, 1}. This language includes strings like 0110, 1001, 11011 and ε, but not 011 or 0101.
- It is easy to verify that the language of palindromes L_{pal} of 0's and 1's is not a regular language.
- To do so we use pumping lemma. If L_{pal} is regular language let n be the associated constant and consider the palindrome $w = 0^n1^n$.
- If L_{pal} is regular then we can break w into w = *xyz* such that y consists of one or more 0's from the first group. Thus *xz* which would also have to be in L_{pal} if L_{pal} were regular would have fewer 0's left of the lone 1 than there are to the right of the 1.
- Therefore *xz* cannot be a palindrome. We have now contradicted the assumption that L_{pal} is regular language.
- There is a natural, recursive definition of when a string of 0's and 1's is in L_{pal}. If start with a basis saying that if a string is a palindrome, it must begin and end with the same symbol.
- Further when the first and last symbols are removed, the resulting string must also be a palindrome. That is:

  Basis: ε, 0 and 1 are palindromes.

  Induction:
- If w is a palindrome, so are 0w0 and 1w1. No string is a palindrome of 0's and 1's unless it follows from this basis and induction rule.

- ✓ A context free grammar is a formal notation for expressing such recursive definitions of languages.
- ✓ A grammar consists of one or more variables that represent classes of strings, i.e. languages.
- ✓ There are rules that say how the strings in each class are constructed. The construction can use symbols of the alphabet, strings that are already known to be in one of the classes or both.

Example:

- ➢ The rules that define the palindromes, expressed in the context free grammar notation are shown in below:
  - ✓ P → ε
  - ✓ P → 0
  - ✓ P → 1
  - ✓ P → 0P0
  - ✓ P → 1P1s
- ➢ Here, the first three rules form the basis. They tell us that the class of palindromes includes the strings ε, 0, and 1. None of the right sides of these rules contains a variable, which is why they form a basis for the definition.

DEFINITION OF CONTEXT FREE GRAMMAR

- ➢ **Formally** context free grammar is a 4-tuple i.e. (***V***, ***Σ***, ***P***, ***S***), where
  - ✓ ***V*** is a finite set called the **variables** also called as **nonterminals** or **syntactic categories.**
    - ▪ Each variable represents a language i.e. set of strings.
  - ✓ ***Σ*** is a finite set of symbols that form the strings of the language being defined. We call this alphabet the **terminals** or **terminal symbols.**
  - ✓ ***P*** is a finite set of **productions** or **rules** that represents the recursive definition of a language. Each production consists of :
    - ▪ A variable that is being (partially) defined by the production. This variable is often called the head of the production.
    - ▪ A production symbol→.
    - ▪ A string of zero or more terminals and variables. This string called the body of the production represents one way to form strings in the language of the variable of the head.
  - ✓ One of the variables (***S***) represents the language being defined; it is called the **start symbol**. Other variables represent the auxiliary classes of strings that are used to help define the language of the start symbol. In our example P the only variable, is the start symbol.

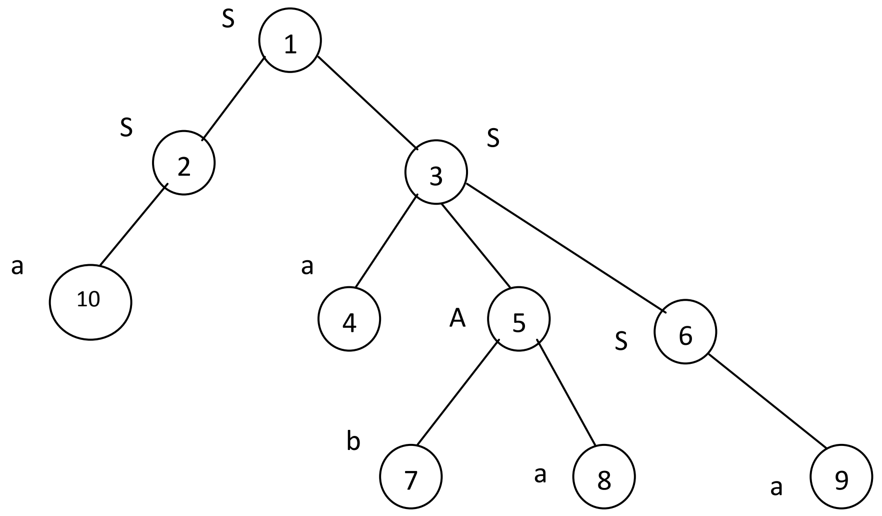
Note: By convention, the start variable is the variable on the left hand side of the first rule.

Example (CFG):

- Consider the grammar given below with the productions as:
  - ✓ $E \rightarrow I$
  - ✓ $E \rightarrow E + E$
  - ✓ $E \rightarrow E * E$
  - ✓ $E \rightarrow (E)$
  - ✓ $I \rightarrow a$
  - ✓ $I \rightarrow b$
  - ✓ $I \rightarrow Ia$
  - ✓ $I \rightarrow Ib$
  - ✓ $I \rightarrow I0$
  - ✓ $I \rightarrow I1$
- The grammar for expressions is stated formally as $G = (\{E, I\}, \Sigma, P, E)$, where Σ is the set of symbols {+, *, (,), a, b, 0, 1} and P is the set of productions shown above.
- Rule 1 is the basis rule for expressions. It says that an expression can be a single identifier.
- Rule 2 through 4 describe the inductive case for expressions. Rule 2 says that an expression can be two expressions connected by a plus sign.
- Rule 3 says the same with a multiplication sign. Rule 4 says that if we take any expression and put matching parenthesis around it, the result is also an expression.
- Rule 5 through 10 describe identifiers I. the basis is rule 5 and 6. They say that a and b are identifiers.
- The remaining four rules are the inductive case. They say that if we have identifier, we can follow it by a, b, 0, or 1, and the result will be another identifier.

DERIVATION TREE

- The derivation in a Context Free Grammar is represented using trees.
- Such trees representing derivation are called as derivation trees.
- A derivation tree (also called as parse tree) for a Context Free Grammar $G = (V, \Sigma, P, S)$ is a tree satisfying the following:
  - ✓ Every vertex has a label which is a variable or terminal or ε.
  - ✓ The root has the label S (i.e. start symbol).
  - ✓ The label of an internal vertex is a variable.
  - ✓ If the vertices $n_1, n_2, \dots, n_k$ written with labels $X_1, X_2, \dots, X_k$ are the sons of the vertex n with label A, then $A \rightarrow X_1X_2 \dots X_k$ is a production in P.
  - ✓ A vertex n is a leaf if its label is $a \in \Sigma$ or ε. n is the only son of its father if its label is ε.
- For example let $G = (\{S, A\}, \{a, b\}, P, S)$, where P consists of $S \rightarrow aAS \mid a \mid SS$, $A \rightarrow SbA \mid ba$. Now the derivation tree is given as follows:



➢ **Note:** In the above diagram

✓ Vertices 4 – 6 are sons of 3 written from left and S → aAS is in P.

✓ Vertices 7 and 8 are sons of 5 written from the left, and A → ba is a production in P.

✓ Vertex 5 is an internal vertex and its label is A, which is a variable.

ORDERING OF LEAVES FROM LEFT

➢ We can order all vertices of a tree in the following way:

✓ The successors of the root (i.e. sons of the root) are ordered from the left.

✓ So vertices at level 1 are ordered from left. If v1 and v2 are any two vertices at level 1 and v1 is to the left of v2 then we say that v1 is to the left of any son of v2.

✓ Also any son of v1 is to the left of v2 and to the left of any son of v2. Thus we get a left – to – right ordering of vertices at level 2.

✓ Repeat this process up to level k, where k is the height of the tree, we have an ordering of all vertices from the left.

➢ In the above diagram the sons of the root are 2 and 3 ordered from the left. So, the son of 2 i.e. "a" is to the left of any son of 3.

➢ The sons of 3 ordered from the left are 4 – 5 – 6. The vertices at level 2 in the left to right ordering are 10 – 4 – 5 – 6.

➢ 4 is to the left of 5. The sons of 5 ordered from the left are 7 – 8. So 4 is to the left of 7. Similarly 8 is to the left of 9.

➢ Thus the order of the leaves from the left is 10 – 4 – 7 – 8 – 9.

➢ **Note:** a sub tree is a one which looks like a derivation tree except that the label of the root may not be S (i.e. start symbol).

SENTENTIAL FORM

➢ Derivations from the start symbol produce strings that have a special role. We call these "sentential forms".

- That is if G = (V, Σ, P, S) is a CFG, then any string α in (V ∪ Σ)* such that $S \overset{*}{\Rightarrow} \alpha$ is a sentential form. It is of two types 1) left sentential form and 2) right sentential form.
- $S \overset{*}{\Rightarrow} \alpha$ is said to be in left sentential form if the leftmost derivation is applied.
- $S \overset{*}{\Rightarrow} \alpha$ is said to be in right sentential form if the rightmost derivation is applied.
- For example consider the sentential form

$$E * (I + E)$$

- Since there is derivation (*from example (CFG)*)

$$E \Rightarrow E * E \Rightarrow E * (E) \Rightarrow E * (E + E) \Rightarrow E * (I + E)$$

- However this derivation is neither leftmost nor rightmost, since at the last step the middle E is replaced.
- As an example of a left sentential form consider ***a * E***, with the leftmost derivation.

$$E \Rightarrow E * E \Rightarrow I * E \Rightarrow a * E$$

- Additionally the derivation

$$E \Rightarrow E * E \Rightarrow E * (E) \Rightarrow E * (E + E)$$

- Shows that ***E * (E + E)*** is a right sentential form.

LEFTMOST DERIVATION OF STRINGS

- A derivation $A \overset{*}{\Rightarrow} w$ is called a leftmost derivation if we apply a production only to the leftmost variable at every step.
- For example consider the productions of the grammar G given below:
  S → aAS | a, A→ SbA | SS | ba
- Now we show how to construct the string aabbaa using leftmost derivation.
  S → aAS → aSbAS → aabAS → aabbaS → aabbaa

RIGHTMOST DERIVATION OF STRINGS

- A derivation $A \overset{*}{\Rightarrow} w$ is called a rightmost derivation if we apply a production only to the rightmost variable at every step.
- For example consider the productions of the grammar G given below:
  S → aAS | a, A→ SbA | SS | ba
- Now we show how to construct the string aabbaa using leftmost derivation.
  S → aAS → aAa → aSbAa → aSbbaa → aabbaa