**Step 5 :** Transition

Draw table for mealy machine

| Present state | Next state | | | |
|---|---|---|---|---|
| | a = 0 | | a = 1 | |
| | State | Output | State | Output |
| → $q_0$ | $q_3$ | 0 | $q_1$ | 1 |
| $q_1$ | $q_1$ | 1 | $q_2$ | 0 |
| $q_2$ | $q_2$ | 0 | $q_3$ | 0 |
| $q_3$ | $q_3$ | 0 | $q_0$ | 0 |

**Difference between Mealy machine and Moore machine**

☞ **Write the difference between Mealy Machine and Moore Machine.**

| Sr. No. | Mealy Machine | Moore Machine |
|---|---|---|
| 1. | Output depends upon present state as well as present input. | Output depends only on present state. |
| 2. | It has less states than moore machine. | However, it has more states than mealy machine. |
| 3. | These machines react faster to inputs. | In these machines more logic is needed to decode the output. Since it has more circuit delays. |

□□□

# CHAPTER 2 Formal Languages

**University Prescribed Syllabus**

**Formal Languages :** Defining Grammar, Derivations, Languges generated by Grammar, Chomsky Classification of Grammar and Languages, Recursive Enumerable Sets, Operations on Languages, Languages and Automata.

## 2.1 Introduction

- The formal languages in automata theory started in 1959, with formal definition given by Roam Chomsky. He tried to find out what is a grammar ? What is a mathematical model of grammar ?
- Let us for take some sentences in English and see how the sentences can be parsed.

**For Example,**

"The man ate the fruit".

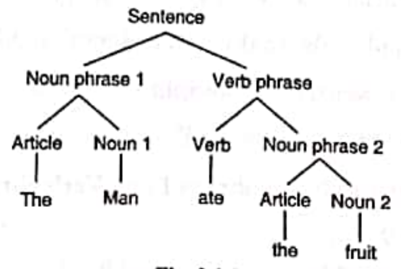

Fig. 2.1.1

- So this is parse tree for this sentence. This parse tree is syntax tree for grammar so it is called as derivation tree.
- In our example, 'Sentence' symbol goes to 'Noun phrase 1' and 'Verb phrase', 'Noun phrase 1' goes to 'Article' and 'Noun 1'. 'Article' goes to 'the' and 'Noun 1' goes to 'man'.
- 'Verb phrase' goes to 'Verb' and 'Noun phrase 2'. 'Noun phrase 2' goes to 'Article' and 'Noun 2'. 'Noun phrase 2' goes to article and 'noun 2'. 'Noun 2' goes to 'fruit'.
- Similarly, 'Verb' goes to 'ate', again 'Article' goes to 'the' which is already there.
- So these are called **rules** or **production** or **production rule**. Using these rules you can desire the sentence.

These rules are,

$$<S> \Rightarrow <NP_1><VP>$$
$$<NP_1> \Rightarrow <Article><N_1>$$
$$<Article> \Rightarrow \text{the}$$
$$<N_1> \Rightarrow \text{man}$$
$$<VP> \Rightarrow <Verb><NP_2>$$
$$<Verb> \Rightarrow \text{ate}$$
$$<NP_2> \Rightarrow <article><N_2>$$
$$<N_2> \Rightarrow \text{fruit}$$

(i) Every derivation start with symbol S.

(ii) When we apply rule, that is written by a double arrow.

(iii) Left side is rewritten as the right hand side.

$$<S> \Rightarrow <NP_1><VP>$$

- Here S is rewritten as Noun phrase 1 and Verb phrase, this double arrow represents directly.
- So S directly derives 'Noun phrase 1' and 'Verb phrase' remains there.

$$<S> \Rightarrow <Article><N_1><VP>$$

- 'Noun phrase 1' directly derives 'Article' and 'Noun'. When we get something from something else, this is called as **sentential form.**
- In this, sentential form a 'Noun phrase 1' is rewritten as 'Article' and 'Noun 1'.
- Apply only one rule for one step :

$$\begin{aligned} <S> &\Rightarrow <Article><N_1><VP> \\ &\Rightarrow \text{The} <N_1><VP> \\ &\Rightarrow \text{The man} <VP> \\ &\Rightarrow \text{The man} <V><NP_2> \\ &\Rightarrow \text{The man ate} <NP_2> \\ &\Rightarrow \text{The man ate} <Article><N_2> \\ &\Rightarrow \text{The man ate the} <N_2> \\ &\Rightarrow \text{The man ate the fruit} \end{aligned}$$

## 2.1.1 Terminals and Non Terminal

☞ **Define the term Terminals and Non Terminals.**

### (a) Terminals

Exam
- **Terminal Symbols** are literal symbols which may appear in the outputs of production rules of formal grammar and which cannot be changed using the rules of the grammar.
- In our previous example, the, man, ate, fruit; they cannot be rewritten as something else, the derivation terminates there, so these are called **terminals.**
- Set of terminals are denoted by 'T'
- Terminals are placed on right hand side in production rule.

(b) **Non terminals**

E-xam

- **Non terminal symbols** are those symbols which can be replaced by groups of terminal symbols according to production rules.
- In our example, 'NP$_1$' is replaced by 'Article' and 'Article' is replaced by 'the'; so the 'NP$_1$' and 'Article' are non terminals and so on.
- [Set of non terminal symbols are denoted by 'N'.]
- [Non terminal symbols are placed on left hand side in production rules.]

**Syllabus Topic : Defining Grammar**

E-xam

### 2.1.2 Grammar

☞ **Define Grammar.**

☞ **Write components of grammar explain it with example.**

A grammar consists of 4 tuples,

G = (N, T, P, S)

N = Finite set of non terminals

* T = Finite set of terminals

P = Finite set of production or production rule

S = Start symbol ∴ $S \in N$

**For example,**

1. Grammar $G_1 = (\{S, A, B\}, \{a, b\}, S, \{S \rightarrow AB, A \rightarrow a, B \rightarrow b\})$

Here,

Non terminal = S, A, B

Terminal symbol = a, b

S = start symbol, $S \in N$

Production P :

$S \rightarrow AB, A \rightarrow a, B \rightarrow$ a. b

2. Grammar $G_1 = (\{S, A\}, \{a, b\}, S, \{S \rightarrow aAb, aA \rightarrow aaA, A \rightarrow \varepsilon\})$

Here,

Non terminal = S, A

Terminal symbol = a, b

S = start symbol, $S \in N$

Production P :

$S \rightarrow aAb$

$aA \rightarrow aaAb$

$A \rightarrow \varepsilon$

**Syllabus Topic : Derivations, Languages Generated by Grammar**

## 2.2 Derivations from Grammar and Languages Generated by Grammar

### 2.2.1 Languages Generated by Grammar

☞ **How to generate language by grammar explain it with example ?**

E-xam

[Set of all strings that can be derived from grammar is said to be **Language** generated from that grammar.]

**For example,**

1. Consider the grammar

$G_1 = (\{S, A\}, \{a, b\}, S, \{S \rightarrow aAb, aA \rightarrow aaAb, A \rightarrow \varepsilon\})$

$S \rightarrow \underline{aAb}$ ...[By production rule $S \rightarrow aAb$]

$\rightarrow a\underline{aA}bb$ ...[$aA \rightarrow aaAb$]

$\rightarrow aaa\underline{A}bbb$ ...[$aA \rightarrow aaAb$]

$\rightarrow aaabbb$ ... [by $A \rightarrow \varepsilon$]

Final string aaabbb is derived from this grammar.

2.

$G_2 = (\{S, A, B\}, \{a, b\}, S, \{S \rightarrow AB, A \rightarrow a, B \rightarrow b\})$

$S \rightarrow AB$

$\rightarrow ab$

$L(G_2) = \{a\ b\}$ ...Language generated by this grammar

3. $G_3 = (\{S, A, B\}, \{a, b\}, S, \{S \rightarrow AB, A \rightarrow aA \mid a, B \rightarrow b\,B \mid b\})$

$S \rightarrow \underline{A}B$

$\rightarrow ab$

$S \rightarrow \underline{A}B$ ...[by $A \rightarrow aA$, $B \rightarrow bB$]

$\rightarrow aAbB$ ...[by $A \rightarrow a$, $B \rightarrow b$]

$\rightarrow aabb$

$S \rightarrow AB$ ...[by $A \rightarrow aA$; $B \rightarrow b$]

$\rightarrow aAb$ ...[$B \rightarrow b$]

$\rightarrow aab$

Depending upon choice we can generate many strings.

$L(G_3) = \{aa, a^2b^2, a^2b \ldots\}$

Generalise language

$L(G_3) = \{a^m b^n \mid m \geq 0 \text{ and } n \geq 0\}$

exam

### 2.2.2 Converting Language into Grammar

**How to generate grammar by language explain it with example ?**

Consider some language and convert it into grammar G which produce those language.

**For example,**

1. Lets consider $L = \{a^n b^m \mid n \geq 1 \text{ and } m \geq 0\}$

Language L ($a^n b^m$) generate strings which having any power of n and m

For $a^n = \{a, aa, aaa \ldots\}$

$A \rightarrow aA \mid a$

Consider if $m = 0$

$\therefore\ b^m \Rightarrow b^0 \Rightarrow 1 \Rightarrow \varepsilon$

$b^m = \{\varepsilon, b, bb, bbb \ldots\}$

$B \rightarrow bB \mid \varepsilon$ ...This is grammar for $b^m$

Grammar is;

$S \rightarrow AB$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid \varepsilon$

2. $L = \{a^n b^m \mid n \geq m\}$

$L(a) = \{ab, aab, aabb, aaabb \ldots\}$

Generate more number of a than b.

$A \rightarrow aA \mid \varepsilon$

Generate equal number of a and b.

$S \rightarrow a\,Ab \mid asb$

$A \rightarrow aA \mid \varepsilon$

**Syllabus Topic : Chomsky Classification of Grammar and Languages**

## 2.3 Chomsky Classification of Languages

### 2.3.1 Classification of Grammar

**Explain Chomsky Classification of grammar.**

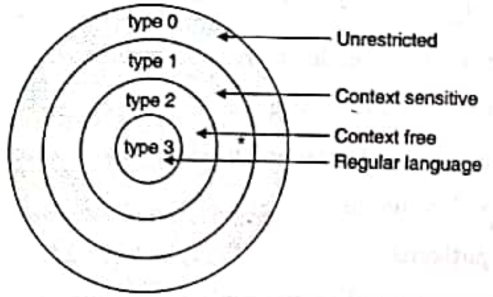

Fig. 2.3.1

- Grammar is classified by considering the form of production.
- They are classified into four types in term of production. **(type 0, type 1, type 2 and type 3).**
- Classification provides a basic for understanding the relationship between grammars.

| Class | Grammars | Languages | Automaton |
|---|---|---|---|
| type 0 | Unrestricted | Recursive enumerable | Turing machine |
| type 1 | Context sensitive | Context sensitive | Linear bound |
| type 2 | Context free | Context free | Pushdown |
| type 3 | Regular | Regular | Finite |

- Type 0 is unrestricted grammar and Type 3 is more restricted grammar

**Note :** Kleen star : $\Sigma^* \rightarrow$ set of all strings

Kleen plus : $\Sigma^+ \rightarrow \Sigma^* - \varepsilon$

($\Sigma^+$ is positive clousure)

**Type 3 : Regular grammar**

☞ **Explain type 3 Grammar.**

☞ **Which machine is used for type 3 ?**

☞ **Which language is used for type 3 ?**

- Type 3 grammar is also called as regular grammar.
- It generates **regular language.**
- Language is generated by using **finite state automaton machine.**
- Use of Regular languages
  1) Search patterns
  2) Lexical structure of programming language.

Regular grammar is divided into two types :

1. Right linear grammar
2. Left linear grammar

E-xam

**1. Right linear grammar**

☞ **Explain Right linear grammar.**

- A grammar is said to be **right linear** if all production are of the form,

$$\underline{A \rightarrow xB}$$
$$A \rightarrow x$$

Where A, B ∈ N and x ∈ T

N : Non terminal

T : Terminal

$A \rightarrow xB$

- If non terminal symbols lies to the right of the terminal symbol, then it is said to be a **right linear grammar.**

**For example :** $S \rightarrow abs \mid b$

a, b = terminal, S = non terminal

e-xam

**2. Left linear grammar**

☞ **Explain Left linear grammar.**

- A grammar is said to be **left linear** if all productions are of the form,

$$\underline{A \rightarrow Bx}$$
$$A \rightarrow x$$

Where A, B ∈ N and x ∈ T

- If non terminal symbol lies to the left of terminal symbol, then it is said to be a **left linear grammar.**

✱ Here B is a non terminal and x is terminal.

e.g.

$S \rightarrow Sbb \mid b$

$b$ = terminal, $S$ = Non terminal

Example :

$A \rightarrow \varepsilon$

$A \rightarrow a$

$A \rightarrow abc$

$A \rightarrow B$

$A \rightarrow abcB$

exam

## Type 2 : Context free grammar

- Explain type 2 Grammar.
- Which machine is used for type 2 ?
- Which language is used for type 2 ?

– Type 2 grammar generates **context free language.**

– These are defined by rules of the form $A \rightarrow y$,

where $A \in N$ [Non terminal]

$y \in (T \cup N)^*$ (string of terminals and non terminals)

– These languages are exactly all languages that can be recognized by non **deterministic pushdown automaton.**

– Context free languages are theoretical basis for syntax of most programming languages.

e.g.

$S \rightarrow aS$

$S \rightarrow aSa$

$S \rightarrow aA$

$S \rightarrow aAB$

$A \rightarrow a$

$B \rightarrow b$

## Type 1 : Context sensitive grammar

- Explain type 1 Grammar.
- Which machine is used for type 1 ?
- Which language is used for type 1 ?

– Type 1 grammar generates the **context sensitive languages.**

– These grammars have rule of the form,

$\alpha A \beta \rightarrow \alpha y \beta$

where,

$A$ = Non-terminal

$\alpha, \beta, y \in (T \cup N)^*$ (string of terminals and non terminals)

the string $\alpha$ and $\beta$ may be empty but $y$ must be non empty.

– Languages described by there grammars are exactly all languages that can be recognised by **Linear Bounded Automaton.**

e.g.

$AB \rightarrow AbBC$

$A \rightarrow bcA$

$B \rightarrow b$

## Type 0 : Unrestricted grammar

- Explain type 0 Grammar.
- Which machine is used for type 0 ?
- Which language is used for type 0 ?

– It is also known as **recursively enumerable languages.**

– Type 0 grammar includes all formal grammars.

– The production have no restrictions.

– Unrestricted grammar can be recognized by a **Turing machine.**

– Production rule can be in the form of :

$\alpha \to \beta$

$\alpha$ = string of terminals and non terminals with at least one non terminal, $\alpha$ should not be null or empty.

$\beta$ = string of terminals and non terminal

$\alpha, \beta \in (T \cup N)^*$ (string of terminals and non terminals)

e.g.

$S \to ACaB$

$BC \to aCB$

$CB \to DB$

$aD \to Db$

**Syllabus Topic : Operations on Languages**

### 2.3.2 Operations on Languages

☞ What are different operations performed on languages ?

1. **Union**

$$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$$

It is exactly same line a sets

2. **Concatenation**

$$A \cdot B = \{xy \mid x \in A \text{ and } y \in B\}$$

It means that joining two symbols together.

3. **Star**

$$A^* = \{x_1, x_2, x_3 \ldots x_k \mid k \geq 0 \text{ and } x \in A\}$$

It means take as many symbols that you want from set A and join them together.

**For example,**

$A = \{pq, r\}$, $B = \{t, uv\}$

**Union**

$A \cup B = \{pq, r, t, uv\}$

**Concatenation**

$A \cdot B = \{pqt, pquv, rt, ruv\}$

**Star**

$A^* = \{\varepsilon, pq, r, pqr, rpq, pqpq, rrpqpqpq \ldots\}$

($\because \varepsilon$ = empty string)

**Theorem 1**

– The class of regular languages is closed under **union.**

– This means that if we have two regular languages A and B. If we perform union operation on A and B, then the resultant of **(Please check sentence)**union of A and B are also regular language.

**Theorem 2**

– The class of regular language is closed under **concatenation.**

– This means that it we have two regular languages A and B. If we perform concatenation operation on A and B then resultant of **(Please check sentence)** concatenation of A and B are regular language.

**Syllabus Topic : Languages and Automata**

### 2.4 Languages and Automata

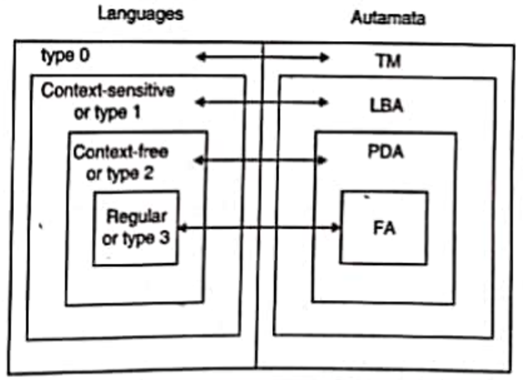

Fig. 2.4.1 : Languages and corresponding automata

| | |
|---|---|
| TM | = Turing machine |
| LBA | = Linear Bounded Automaton |
| PDA | = Pushdown Automation |
| FA | = Finite Automaton |

□□□

# CHAPTER 3 Regular Set and Regular Grammar

**University Prescribed Syllabus**

**Regular Sets and Regular Grammar :** Regular Grammar, Regular Expressions, Finite automata and Regular Expressions, Pumping Lemma and its Applications, Closure Properties, Regular Sets and Regular Grammar.

**Syllabus Topic : Regular Grammar**

## 3.1 Regular Grammar

- A regular grammar is a mathematical object,
- Grammar having four tuples, G = (N, Σ, P, S), where

  N is a nonempty, finite set of non-terminal symbols,
- Σ is a finite set of terminal symbols , or alphabet, symbols,
- P is a set of grammar rules,
- S is the start symbol. Where $S \in N$ (S belong to N)
- Every regular grammar describes a regular language. A **regular grammar** is formal grammar that is right-regular or left-regular.
- A **right regular grammar** is also called **right linear grammar** such that all the production rules in P are in following forms:

$$A \rightarrow a$$
$$A \rightarrow aB$$
$$A \rightarrow \varepsilon$$