THEORY OF AUTOMATA AND FORMAL LANGUAGES

BCSC0011

-By Nishtha Parashar

Assistant Professor, GLA University Mathura

INTRODUCTION

- Theory of Computation is an accumulation of mathematicians work to make a model for a machine that can do thinking and calculations.
- The concept of a machine at early 1900 was devised that does physical work.
- Scientists effort started with a machine that can do specific calculations like encrypting text using specific set of steps.
- Alan Turing believed he could invent a machine that can solve more than one type of problems.

INTRODUCTION

- It started before World War II, Germans army used Enigma encryption.
- Alan Turing and many mathematicians tried to break the Enigma encryption.
- Their efforts resulted in emergence of a mechanical device that was dedicated for deciphering Enigma encrypted messages.
- As a result many German submarines were attacked and destroyed.

INTRODUCTION

- Von Newman, Alan Turing and many others continued working on creating a model for a generic machine that can solve different types of problems.
- The accumulation of their work resulted in emergence of a collection of theorems called **Theory of Computation.**

What is TOC?

- The “Theory of Computation” is a branch of computer science that deals with how efficiently problems can be solved on a model of computation, using an algorithm.
- TOC emerged to give answers for “What are the fundamental capabilities and limitations of computing machines?”
- Most powerful and modern super computers can NOT solve some problems!!
- No matter how much processors get fast, no matter how much memory can be installed; the unsolved problems remain unsolved.

Need of TOC?

- Technologies become obsolete but basic theories remain forever.
- TOC provides tool for solving computational problems like regular expressions for string parsing and pattern matching.
- Studying different types of grammars like CFG would help in many other areas like compiler design and natural language processing.

Branches of TOC

- The field is divided into three major branches:
- Automata theory and language
- Computability theory
- Complexity theory

Branches of TOC

- Automata Theory: mathematical models for computational problems such as pattern recognition and other problems.

Complexity Theory

- The main question asked in this area is “What makes some problems computationally hard and other problems easy?”
- A problem is called “easy”, if it is efficiently solvable.

- Examples of “easy” problems are (i) sorting a sequence of say 1,000,000 numbers, (ii) searching for a name in a telephone directory.

Complexity Theory

- A problem is called “hard”, if it cannot be solved efficiently, or if we don’t know whether it can be solved efficiently.

- Examples of “hard” problems are (i) factoring a 300-digit integer into its prime factors.

Computability Theory

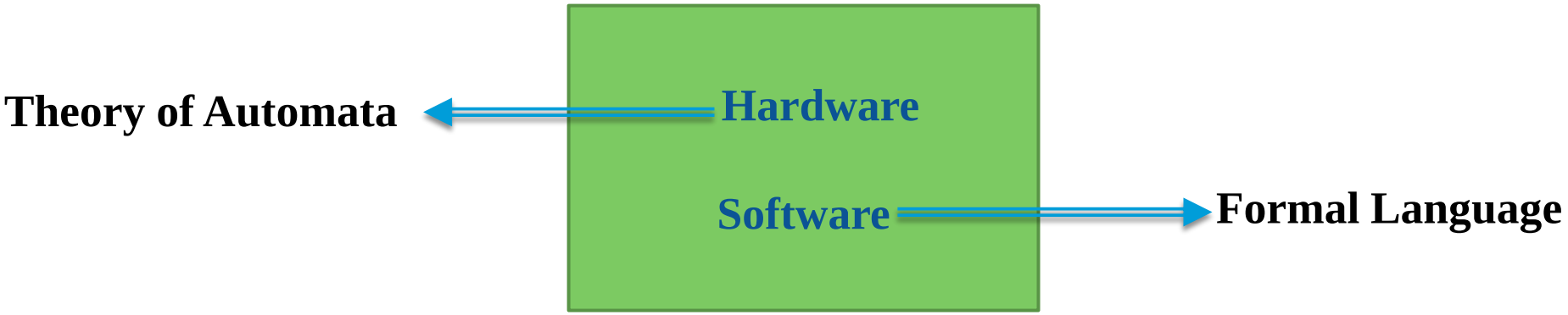
- Computability theory was discovered by Godel, Turing, and Church in 1930's which *states that some of the fundamental problems cannot be solved by a "computer*".
- To attack such a problem, we need formal definitions of the notions of computer, algorithm, and computation.
- The theoretical models that were proposed in order to understand solvable and unsolvable problems led to the development of real computers.

Automata Theory

- Automata Theory deals with definitions and properties of different types of “computation models”. Example of such models:
- Finite Automata: These are used in text processing, compilers, and hardware design.
- Context-Free Grammars: These are used to define programming languages and in Artificial Intelligence.
- Turing Machines: These form a simple abstract model of a “real” computer, such as your PC at home.

Purpose and motivation

- What are the mathematical properties of computer hardware and software?
- What is a computation and what is an algorithm? Can we give mathematical definitions of these notions?
- What are the limitations of computers? Can “everything” be computer?
- **Purpose of TAFL:** Develop formal mathematical models of computation that reflect real-world computers.


Theory of Automata
Hardware
Software
Formal Language

What is the study of Automata Theory?

The study of abstract computing devices, or "machines."

Came before digital computers

What is possible to compute with an abstract machine - Seminal work by Alan Turing

Why is this useful?

- Direct application to creating compilers, programming languages, designing applications.
- Formal framework to analyze new types of computing devices, e.g. bio computers or quantum computers.
- ***Develop mathematically mature computer scientists capable of precise and formal reasoning!***

Theory of Automata

-Study of Abstract Computing devices.

- Finite Automata (1943) by McCulloch & Walterpitts
- Pushdown Automata (1967) by Ginsburg & Greibach
- Linear Bounded Automata (1960) by John Myhill
- Turing Machine (1936) by Alan Turing

- FA<PDA<LBA<TM

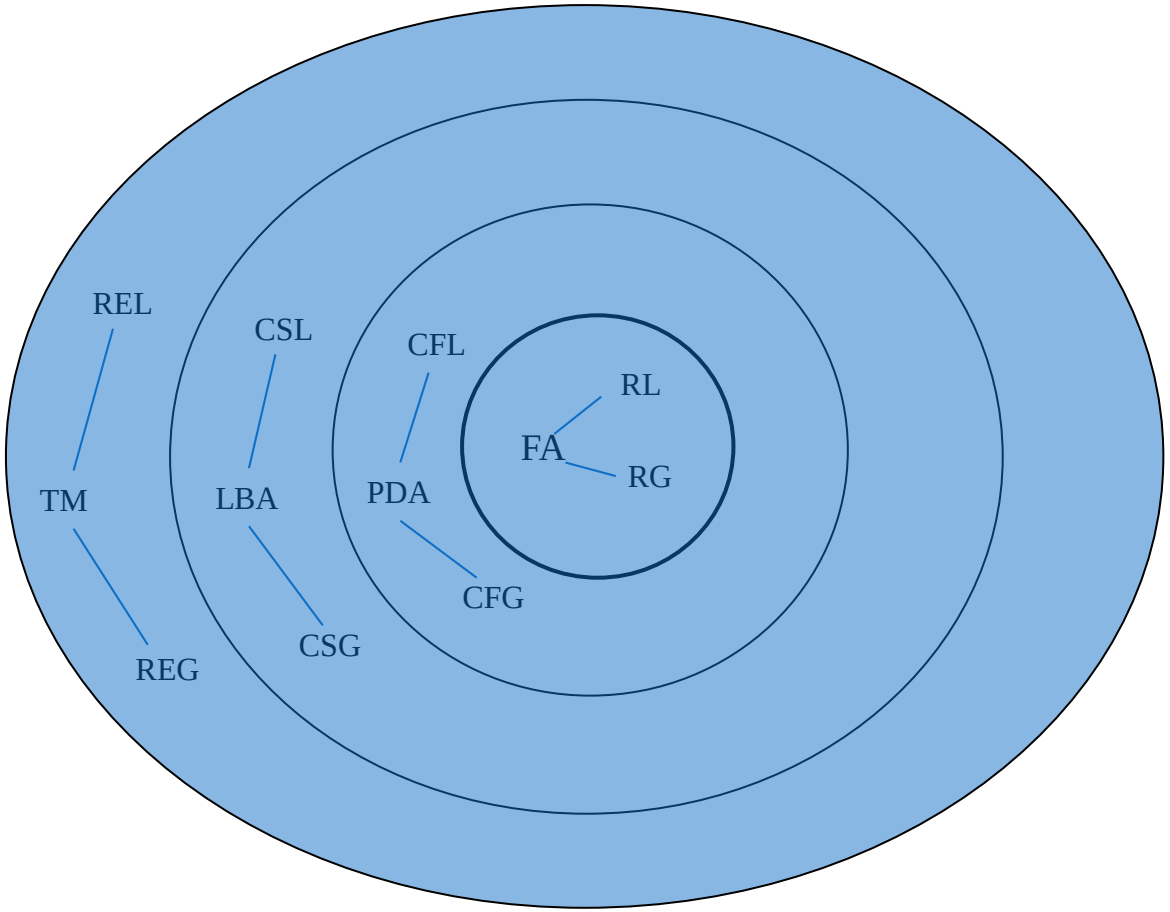
Formal Language

-Study of Abstract programming language.

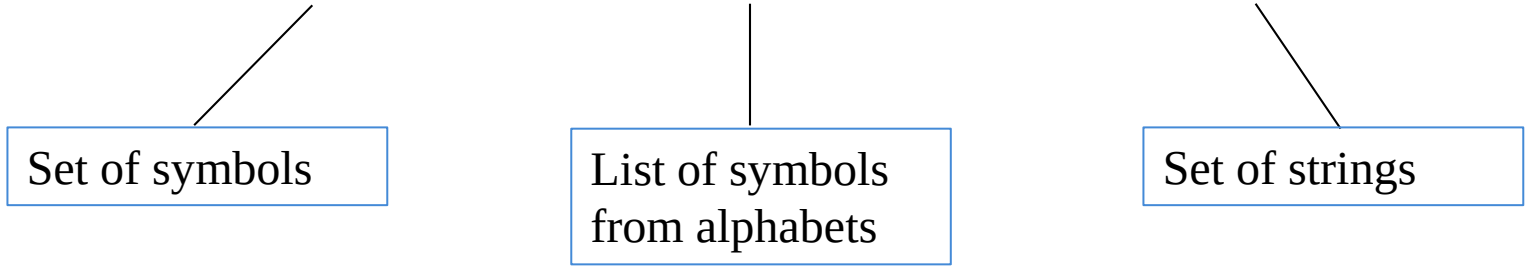
- Regular language & Regular grammar
- Context Free language and Context Free grammar
- Context-Sensitive language and Context-Sensitive grammar
- Recursive Enumerable language & grammar.

- By Noam Chomsky 1956.
- A grammar is a set of production rules which are used to generate strings of a language.

Formal Languages



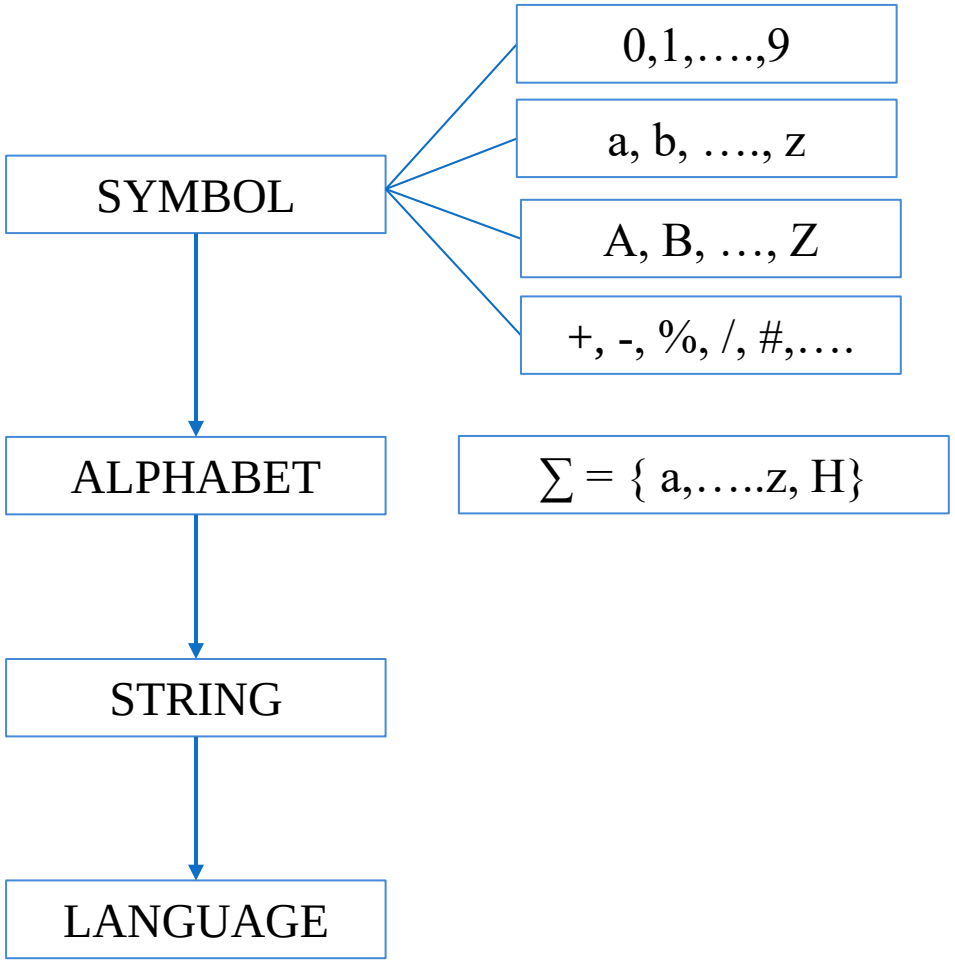
ALPHABETS, STRING AND LANGUAGE

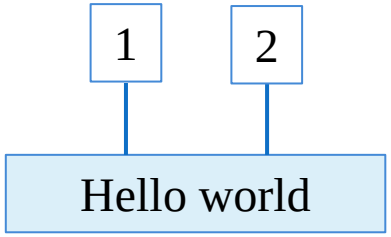


Alphabets: An alphabet is a finite, non-empty set of symbols, it is denoted by $\sum$

String: A string is a finite sequence of symbol chosen from an alphabet.

Language: A set of strings all of which are chosen from some $\sum^*$


0,1,….,9
a, b, …., z
A, B, …, Z
+, -, %, /, #,….
SYMBOL
ALPHABET
∑ = { a,…..z, H}
STRING
LANGUAGE


1
2
Hello world

Alphabet

An alphabet is a **finite**, **non-empty** set of symbols, it is denoted by ∑

∑={0, 1}

∑={a, e, i, o, u}

Numbers: 0, 1, ….., 9

Characters: a, b, ……. z ; A, B, ……Z

Special Symbols: +, -, /, %, #.....

String

A string is a **finite** sequence of symbol chosen from an alphabet.

1. 00111 is a string from $\sum=\{0, 1\}$
2. ‘automata’ is a string from $\sum = \{ a,\ldots..z\}$

Basic Terminologies

1. Empty string
2. Length of a string
3. Power of an alphabet
4. Concatenation of strings

1. Empty string(∧): No string is generated.
2. Length of a string: Total no. of characters in a string.

   $|0011|=4$; $|automata|=8$; $|\wedge|=0$

3. Power of an alphabet: Represented by $\sum^k$

   $\sum=\{a, b, c\}$

   $\sum^1=\{a, b, c\} = 3^1 = 3$

   $\sum^2=\{aa, bb, cc, ab, ba, ac, ca, bc, cb\} = 3^2 = 9$

   $\sum^3=\{aaa, aab, aba, abc, baa, bcc,\ldots\ldots\ldots\}= 3^3 = 27$

String

4. Kleen Star Closure ($\sum^*$) = Set of all strings.

   $\sum^* = \sum^0 U \sum^1 U \sum^2 U \sum^3 U \ldots\ldots\ldots \sum^n$

5. Concatenation of a string

   X=0011 ; Y=0101

   XY=00110101

   YX=01010011

   $XY \neq YX$

Language

A set of strings all of which are chosen from some $\sum^*$

$L \subset \sum^*$

$L \subset \sum^1 U \sum^1 U \sum^2 U \sum^3$

Q. Find $\sum^*$ for $\sum=\{a, b, c\}$

$\sum^* = \sum^0 U \sum^1 U \sum^2 U \sum^3$

where,

$\sum^1=\{a, b, c\} = 3^1 = 3$

$\sum^2=\{aa, bb, cc, ab, ba, ac, ca, bc, cb\} = 3^2 = 9$

$\sum^3=\{aaa, aab, aba, abc, baa, bcc,\ldots\ldots\ldots\ldots\}= 3^3 = 27$

$\sum^* =\{a, b, c, aa, bb, cc, ab, ba, ac, bc, cb, aaa, aab, aba,\ldots\ldots\}= 39$

Language

$\{a, bb\} \subset \{a, b, c, aa, bb, cc, ab, ba, ac, bc, cb, aaa, aab, aba, \ldots\ldots\}$

$\{ab, ba\} \subset \{a, b, c, aa, bb, cc, ab, ba, ac, bc, cb, aaa, aab, aba, \ldots\ldots\}$

Language (subset of $\sum^*$)

Language

Ques 1. The language of string consisting of n 0's followed by n is for some n>=0 and ∑={0, 1}

L={∧, 01, 0011, 000111, …..}

Ques 2. Find set of binary numbers whose value is prime.

L={ 10, 11, 101, 111,……}

Exercise 1

Assume $\sum = \{0,1\}$

1. **How many elements are there in $\sum^2$?**

   Length($\sum$) x Length($\sum$)=2x2=2^2 =4

2. **How many combinations of $\sum^3$?**

   2x2x2= 2^3

3. **How many elements are there in $\sum^n$?**

   (Length($\sum$))^n =2^n

Exercise 2

Assume A1 = {AM, PM}, A2= {1,2,……59}, A3={1,2,….12}

1. **How many elements are there in A1 A3 ?**

   Length (A1) x Length (A3) = 2x12=24

2. **How many elements are there in A1 A2 A3 ?**

   Length (A1) x Length (A2) x Length (A3) = 2 x 59 x 12 = 1416

Exercise 3

Assume HexaDecimal=

{0,1,…. 9,A,B,C,D,E,F}

1. **How many HexaDecimals of length 4?**

   16^4

2. **How many HexaDecimals of length n?**

   16

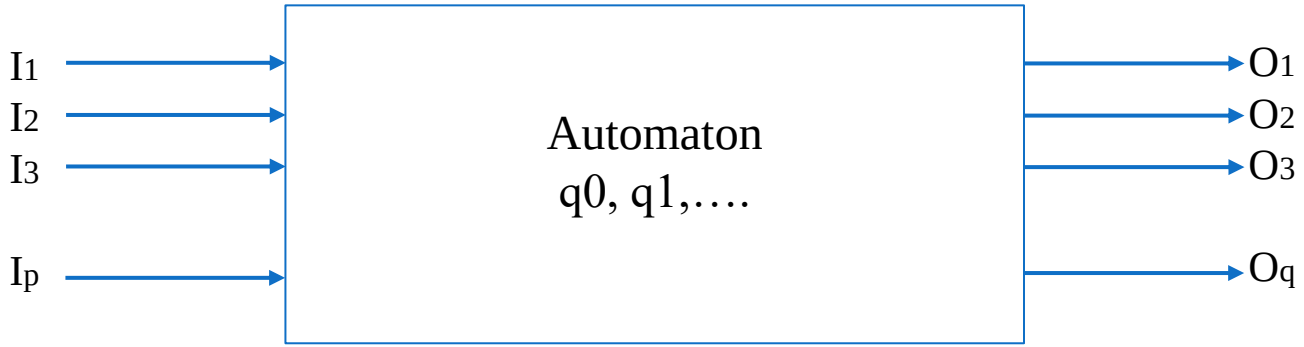
3. **How many elements are there in $\{0,1\}^8$?**

   $2^8 = 256$

AUTOMATA AND GRAMMARS

Automata- An Automaton is defined as a system where energy, materials and information are transferred, transmitted and used for performing some function *without direct participation of human*.

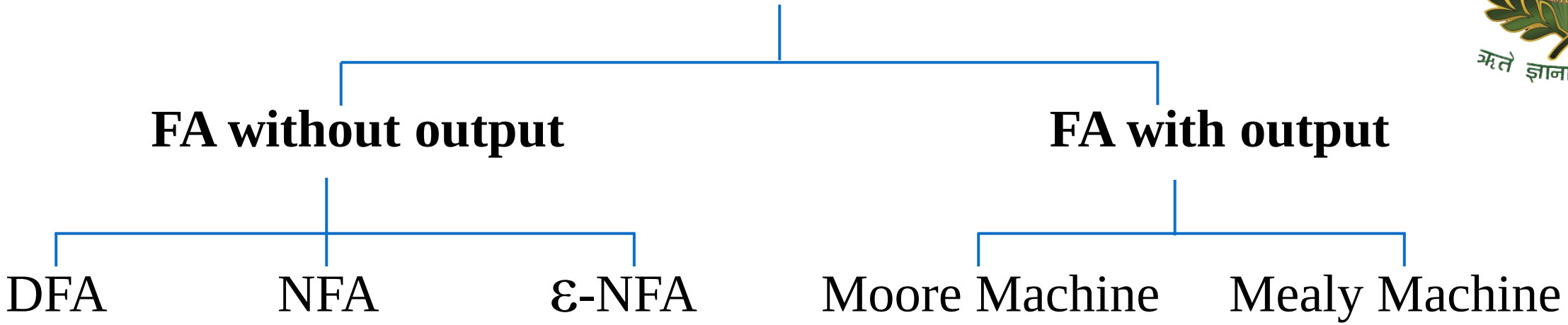
E.g. Automatic Printing Machine, Automatic packing machine etc.



CHARACTERISTICS OF AUTOMATA

1. Input – Finite number of input from $\sum$ e.g. I_1, I_2,….. I_p
2. Output – Finite number of output eg. O_1, O_2,……., O_q
3. State – At any instant of time automaton can be in one of the state.
4. State Relation – Relation between present input and present state.
5. Output Relation – Either state only or both state and input.

FINITE AUTOMATA



FA: A Finite Automata has a set of states and its ‘control move’ from state to state in response to external ‘input’.

Output Depends on-

Input	State	Machine	E.g.
Yes	No	Automata without memory	DFA, NFA
Yes (overall)	Yes	With memory	PDA, LBA, TM
No	Yes	Moore Machine	
Yes (At any instant of time)	Yes	Mealy Machine	

GRAMMARS

GRAMMAR: The mathematical model of grammar was given by Noam Chomsky, it turn out to be useful for computer language.

A Grammar is defined as **4-tuple G= (V, ∑, P, S)**

1. V → Finite non-empty set whose elements are called **non-terminal**.
2. ∑ → Finite non-empty set whose elements are called **terminal**.
3. $V \cap \sum = \phi$
4. S is a special **non-terminal** ($S \in V$) is start symbol.
5. P is set of **production rules** e.g. $\alpha \rightarrow \beta$ where α has at least one non-terminal.

GRAMMARS

Note:

1. Reverse substitution is not permitted. E.g. S → AB
2. No inversion operation is permitted e.g. if S→AB then no production AB→S

G=({S,A,B,E}, {a,b,c}, P, S) **[G= (V, ∑, P, S)]**

P:

S → AB

A → a

B → b

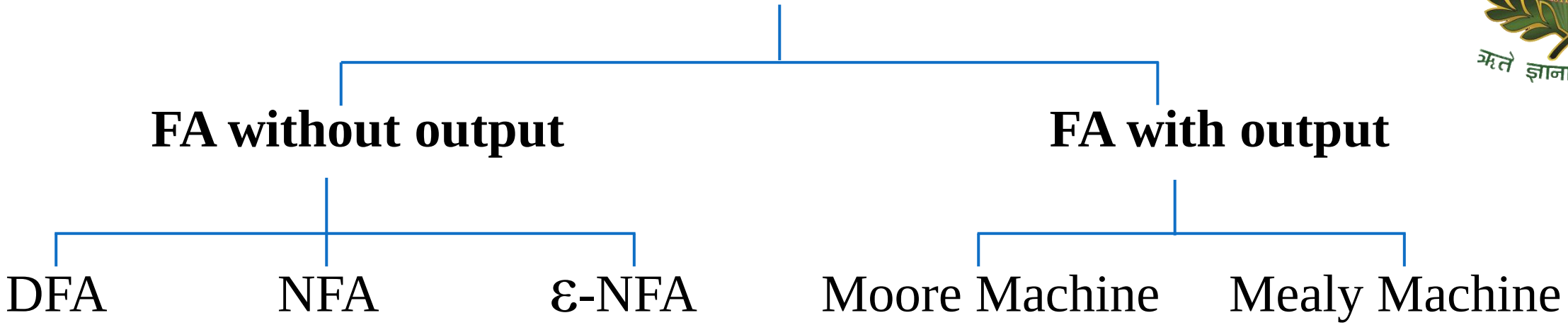
E → c

Chomsky Hierarchy

Grammar Type	Type 0	Type 1	Type 2	Type 3
Grammar Accepted	Unrestricted Grammar	Context Sensitive Grammar	Context Free Grammar	Regular Grammar
Language Accepted	Recursively Enumerable	Context Sensitive	Context Free	Regular Language
Automaton	Turing Machine	Linear Bounded Automata	Pushdown Automata	Finite State Automata
Example	$\alpha \rightarrow \beta$ **where,** **α is $(\sum \cup V)^*V(\sum \cup V)^*$** **$\beta \in (\sum \cup V)^*$**	**$\alpha A \beta \rightarrow \alpha\gamma\beta$** **where,** **$A \in V$** **$\alpha\beta\gamma \in (\sum \cup V)^*$** **$\alpha\beta$ can be empty** **γ is non empty**	**$A \rightarrow \gamma$** **(Left side has only 1 variable)** **where,** **$A \in V$** **$\gamma \in (\sum \cup V)^*$**	**$X \rightarrow a \mid aY$ (Right Linear)** **$X \rightarrow a \mid Ya$ (Left Linear)** **where, $X,Y \in V$** **$a \in \sum$** **$S \rightarrow \wedge$ is allowed if it does not appear on the right side of any rule**

Grammar Type	Type 0	Type 1	Type 2	Type 3
Grammar Accepted	Unrestricted Grammar	Context Sensitive Grammar	Context Free Grammar	Regular Grammar
Language Accepted	Recursively Enumerable	Context Sensitive	Context Free	Regular Language
Automaton	Turing Machine	Linear Bounded Automata	Pushdown Automata	Finite State Automata
Example	**$\alpha \rightarrow \beta$** **where,** **α is $(\sum \cup V)^* V (\sum \cup V)^*$** **$\beta \in (\sum \cup V)^*$**	**$\alpha A \beta \rightarrow \alpha\gamma\beta$** **where,** **$A \in V$** **$\alpha\beta\gamma \in (\sum \cup V)^*$** **$\alpha\beta$ can be empty** **γ is non empty**	**$A \rightarrow \gamma$** **(Left side has only 1 variable)** **where,** **$A \in V$** **$\gamma \in (\sum \cup V)^*$**	**$X \rightarrow a \mid aY$ (Right Linear)** **$X \rightarrow a \mid Ya$ (Left Linear)** **where, $X, Y \in V$** **$a \in \sum$** **$S \rightarrow \wedge$ is allowed if it does not appear on the right side of any rule**
Example	$S \rightarrow ACaB$ $Bc \rightarrow acB$ $CB \rightarrow DB$ $aD \rightarrow Db$	$AB \rightarrow AbBc$ $A \rightarrow bcA$ $B \rightarrow b$	$S \rightarrow X\,a$ $X \rightarrow a$ $X \rightarrow aX$ $X \rightarrow abc$ $X \rightarrow \varepsilon$	$X \rightarrow \varepsilon$ $X \rightarrow a \mid aY$ $Y \rightarrow b$

FINITE AUTOMATA



FA: A Finite Automata has a set of states and its ‘control move’ from state to state in response to external ‘input’.

DETERMINISTIC FINITE AUTOMATA (DFA)

In DFA on each input there is only one state which the automata can transition from its current state.

A DFA consist of 5-tuple (Q, ∑, δ, q0, F)

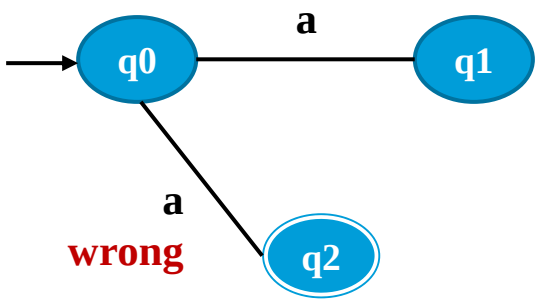
Q→ Finite non-empty set of state.

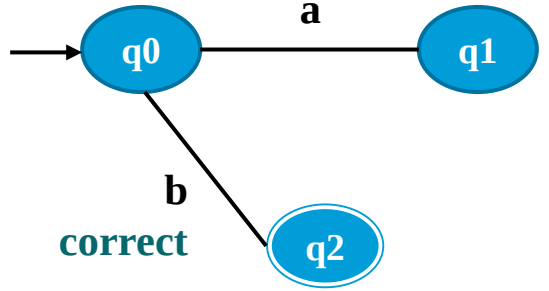
∑ → Finite non-empty set of input.

δ → Transition Function Q x ∑ → Q [e.g. δ(q0,a)→q1]

q0 → q0 ε Q is initial state.

F ⊆ Q is a set of final states called acceptance state.



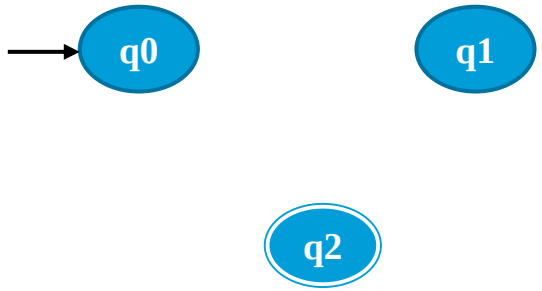


DETERMINISTIC FINITE AUTOMATA (DFA)

In DFA on each input there is only one state which the automata can transition from its current state.

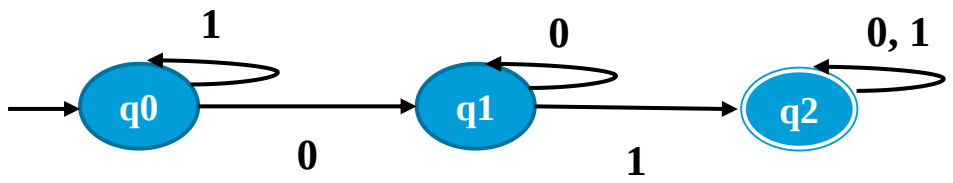
δ → Transition Function Q x ∑ → Q

For,
Q={q0, q1, q2}
∑={a, b}
Total no. of transition functions= 3x2 → 3



DETERMINISTIC FINITE AUTOMATA (DFA)

Construct DFA that accepts all possible set of strings containing 01, given ∑={0, 1}



A DFA consist of 5-tuple (Q, ∑, δ, q0, F)
Q={ q0, q1, q2}
∑={0,1}
F={q2}
δ : Transition Function
Q x ∑ → Q

δ : Q x ∑ → Q
δ =
δ(q0,1) → q0
δ(q0,0) → q1
δ(q1,0) → q1
δ(q1,1) → q2
δ(q2,0) → q2
δ(q2,1) → q2

Thank You

Definitions

∑ **(Alphabet):** a finite set of letters, denoted by ∑

Letter: an element of an alphabet ∑

Word: a finite sequence of letters from the alphabet ∑

∧**(empty string):** a word without letters.

Language ∑* (Kleen's Star): the set of all words on ∑