Theory of Automata and Formal Languages ACSE0404

Unit:1

Finite Automata

Sana Anjum
Assistant Professor
Computer Science and Engineering

B. Tech
(Computer Science & Engineering)
4th Semester

Faculty Profile

•I Shruti Sinha currently working as Assistant Professor in NIET Greater Noida having experience of 1year and 8 months in teaching.

•I have completed my Btech(CSE) as well as Mtech(AI) from Gautam Buddha University,Greater Noida in 2020.

•My area of research is Artifial Intelligence ,Machine Learning.

NIET Greater Noida

GET FUTURE READY

AN AUTONOMOUS INSTITUTE

Evaluation Scheme

NOIDA INSTITUTE OF ENGINEERING & TECHNOLOGY, GREATER NOIDA
(An Autonomous Institute)

B. TECH (IT)
EVALUATION SCHEME
SEMESTER-IV

Subject Codes	Subject Name	Periods			Evaluation Scheme				End Semester		Total	Credit
		L	T	P	CT	TA	TOTAL	PS	TE	PE		
AAS0402	Engineering Mathematics-IV	3	1	0	30	20	50		100		150	4
AASL0401	Technical Communication	2	1	0	30	20	50		100		150	3
AIT0401	Software Engineering	3	0	0	30	20	50		100		150	3
ACSE0403A	Operating Systems	3	0	0	30	20	50		100		150	3
ACSE0404	Theory of Automata and Formal Languages	3	0	0	30	20	50		100		150	3
ACSAI0402	Database Management Systems	3	1	0	30	20	50		100		150	4
AIT0451	Software Engineering Lab	0	0	2				25		25	50	1
ACSE0453A	Operating Systems Lab	0	0	2				25		25	50	1
ACSAI0452	Database Management Systems Lab	0	0	2				25		25	50	1
ACSE0459	Mini Project using Open Technology	0	0	2				50			50	1
ANC0402 / ANC0401	Environmental Science*/Cyber Security*(Non Credit)	2	0	0	30	20	50		50		100	0
	MOOCs** (For B.Tech. Hons. Degree)											
	GRAND TOTAL										**1100**	**24**

Syllabus

Course Contents / Syllabus		
UNIT-I	**Basic Concepts of Formal Language and Automata Theory**	**8 Hours**
Introduction to Theory of Computation- Alphabet, Symbol, String, Formal Languages, Grammar, Derivation and Language generation by Grammar, Chomsky Hierarchy, Finite Automata, Deterministic Finite Automaton (DFA)- Definition, Representation, Acceptability of a String and Language, Non-Deterministic Finite Automaton (NFA), Equivalence of DFA and NFA, NFA with ∈-Transition, Equivalence of NFA's with and without ∈-Transition, Finite Automata with output- Moore Machine, Mealy Machine, Equivalence of Moore and Mealy Machine, Minimization of Finite Automata, Myhill-Nerode Theorem, Simulation of DFA and NFA.		
UNIT-II	**Regular Language and Finite Automata**	**8 Hours**
Regular Expressions, Transition Graph, Kleen's Theorem, Finite Automata and Regular Expression- Arden's theorem, Algebraic Method Using Arden's Theorem, Regular Grammars-Right Linear and Left Linear grammars, Conversion of FA into Regular grammar and Regular grammar into FA, Regular and Non-Regular Languages- Closure properties of Regular Languages, Pigeonhole Principle, Pumping Lemma, Application of Pumping Lemma. Decidability- Decision properties, Finite Automata and Regular Languages, Simulation of Transition Graph and Regular language.		
UNIT-III	**Context Free Language and Grammar**	**8 Hours**
Context Free Grammar (CFG)-Definition, Derivations, Languages, Derivation Trees and Ambiguity, Simplification of CFG, Normal Forms- Chomsky Normal Form (CNF), Greibach Normal Form (GNF), Pumping Lemma for CFL, Closure properties of CFL, Decision Properties of CFL		
UNIT-IV	**Push Down Automata**	**8 Hours**
Pushdown Automata- Definition, Representation, Instantaneous Description (ID), Acceptance by PDA, Nondeterministic Pushdown Automata (NPDA)- Definition, Moves, Pushdown Automata and Context Free Language, Pushdown Automata and Context Free Grammar, Two stack Pushdown Automata.		
UNIT-V	**Turing Machine and Undecidability**	**8 Hours**
Turing Machine Model, Representation of Turing Machines, Language Acceptability of Turing Machines, Techniques for Turing Machine Construction, Variations of Turing Machine, Turing Machine as Computer of Integer Functions, Universal Turing machine, Linear Bounded Automata,		

Branch wise Applications

Computer Science

- Automaton is nothing but a machine which accepts the strings of a language L over an input alphabet Σ. There are four different types of Automata that are mostly used in the theory of computation (TOC).
- Finite-state machine (FSM).
- Pushdown automata (PDA).
- Linear-bounded automata (LBA).
- Turing machine (TM).

Course Objectives

The primary objective of this course is to introduce students to the foundations of computability theory. The other objectives include:

- Introduce concepts in automata theory and theory of computation
- Identify different formal language classes and their relationships
- Design grammars and recognizers for different formal languages
- Prove or disprove theorems in automata theory using its properties
- Determine the decidability and intractability of computational problems

Course Outcomes

CO	At the end of course , the student will be able to	Bloom's (KL)
CO1	Design and Simplify automata for formal languages and transform non-deterministic finite automata to deterministic finite automata.	K_6
CO2	Identify the equivalence between the regular expression and finite automata and apply closure properties of formal languages to construct finite automata for complex problems.	K_3
CO3	Define grammar for context free languages and use pumping lemma to disprove a formal language being context- free.	K_3
CO4	Design pushdown automata (PDA) for context free languages and Transform the PDA to context free grammar and vice-versa.	K_6
CO5	Construct Turing Machine for recursive and recursive enumerable languages. Identify the decidable and undecidable problems.	K_6

Program Outcomes (POs)

Engineering Graduates will be able to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

Program Outcomes (POs)

Contd..

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

Program Outcomes (POs)

Contd..

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

CO-PO correlation matrix

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	2	2	3	3	2	2	-	-	2	1	-	3
CO2	1	3	2	3	2	2	-	1	1	1	2	2
CO3	2	2	3	2	2	2	-	2	2	1	2	3
CO4	2	2	2	3	2	2	-	2	2	1	1	3
CO5	3	2	2	2	2	2	-	2	1	1	1	2
Average	2	2.2	2.4	2.6	2	2	-	1.4	1.6	1	1.2	2.6

CO-PO correlation matrix

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	2	2	3	3	2	2	-	-	2	1	-	3

Program Specific Outcomes

On successful completion of graduation degree the Computer Science & Engineering graduates will be able to:

PSO1:	identify, analyze real world problems and design their ethical solutions using artificial intelligence, robotics, virtual/augmented reality, data analytics, block chain technology, and cloud computing.
PSO2:	Design and develop the hardware sensor devices and related interfacing software systems for solving complex engineering problems.
PSO3:	understand inter-disciplinary computing techniques and to apply them in the design of advanced computing.
PSO4:	Conduct investigation of complex problem with the help of technical, managerial, leadership qualities, and modern engineering tools provided by industry sponsored laboratories.

CO-PSO correlation matrix

CO	PSO			
	PSO1	**PSO2**	**PSO3**	**PSO4**
CO1	2	2	2	2
CO2	2	2	1	1
CO3	2	2	1	1
CO4	2	2	1	1
CO5	2	2	2	2
Average	2	2	1.4	1.4

Program Educational Objectives

PEO1: To have an excellent scientific and engineering breadth so as to comprehend, analyze, design and solve real-life problems using state-of-the-art technologies.

PEO2: To lead a successful career in industries, to pursue higher studies or to support entrepreneurial endeavors so that engineering graduates can face the global challenges.

PEO3: To effectively bridge the gap between industry and academia through effective communication skill, professional attitude, ethical values and a desire to learn.

PEO4: To provide highly competitive environment and solidarity to students for successful professional career as engineer, scientist, entrepreneur and bureaucrats for the betterment of society.

Result Analysis

Subject Result:

Section A: 95.65%
Section B: 97.14%
Section C: 84.51%
Section D: 94.29%

End Semester Question Paper Template

Printed page: .... Subject Code: ....

Roll No:

NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY ,GREATER NOIDA

(An Autonomous Institute Affiliated to AKTU, Lucknow)

B.Tech/B.Voc./MBA/MCA/M.Tech (Integrated)

(SEM: ..... THEORY EXAMINATION (2020-2021)

Subject ....

Time: 3 Hours Max. Marks:100

General Instructions:

- All questions are compulsory. Answers should be brief and to the point.
- This Question paper consists of ....pages & ...8.........questions.
- It comprises of three Sections, A, B, and C. You are to attempt all the sections.
- Section A -Question No- 1 is objective type questions carrying 1 mark each, Question No- 2 is very short

		SECTION – A		CO
1.	Attempt all parts-		[10×1=10]	
	1-a.	Question-	(1)	
	1-b.	Question-	(1)	
	1-c.	Question-	(1)	
	1-d.	Question-	(1)	
	1-e.	Question-	(1)	
	1-f.	Question-	(1)	
	1-g.	Question-	(1)	
	1-h.	Question-	(1)	
	1-i.	Question-	(1)	
	1-j.	Question-	(1)	

End Semester Question Paper Template

2.	Attempt all parts-		[5×2=10]	CO
	2-a.	Question-	(2)	
	2-b.	Question-	(2)	
	2-c.	Question-	(2)	
	2-d.	Question-	(2)	
	2-e.	Question-	(2)	
SECTION – B				CO
3.	Answer any five of the following-		[5×6=30]	
	3-a.	Question-	(6)	
	3-b.	Question-	(6)	
	3-c.	Question-	(6)	
	3-d.	Question-	(6)	
	3-e.	Question-	(6)	
	3-f.	Question-	(6)	
	3-g.	Question-	(6)	

End Semester Question Paper Template

SECTION – C				CO
4	Answer any one of the following-		[5×10=50]	
	4-a.	Question-	(10)	
	4-b.	Question-	(10)	
5.	Answer any one of the following-			
	5-a.	Question-	(10)	
	5-b.	Question-	(10)	
6.	Answer any one of the following-			
	6-a.	Question-	(10)	
	6-b.	Question-	(10)	
7.	Answer any one of the following-			
	7-a.	Question-	(10)	
	7-b.	Question-	(10)	
8.	Answer any one of the following-			
	8-a.	Question-	(10)	
	8-b.	Question-	(10)	

Prerequisite for the Course

- Basics operations of mathematics.
- Discrete mathematics.
- Predicate logic.

Recap

- Finite automata is an abstract computing device. It is a mathematical model of a system with discrete inputs, outputs, states and a set of transitions from state to state that occurs on input symbols from the alphabet Σ.

Brief subject introduction with Video Lecture

- [(73) TAFL1:Theory of Automata and Formal Language| Course Overview of automata, TOC Lectures in Hindi - YouTube]
- [(73) TAFL2:Theory of Automata and Formal Language| Theory of Computation Tutorial Syllabus - YouTube]

Unit Contents

Topics	Duration (in Hours)
Introduction to theory of computation	1
Finite Automata	7
Minimization of DFA	1
Finite Automata with Output	2

Topics	Duration (in Hours)
Assignment, Tutorials and Quiz	1

Unit Objectives

Objective of the unit is to make students able to:

- Construct DFA and NFA.
- Apply the uses of FA in computational problems.
- Minimize the Finite automata.
- Simulate NFA and DFA.

Objective of the Topic

The objective of the topic is to make the student able to:

- Understand the requirement of finite automata.
- Implement Finite automata
- Realize the expressive power of Ɛ- NFA, NFA and DFA
- Minimize the FA
- Implement Mealy and Moore machine.
- Convert Mealy machine to Moore machine and vice-versa.

Objective mapping with CO

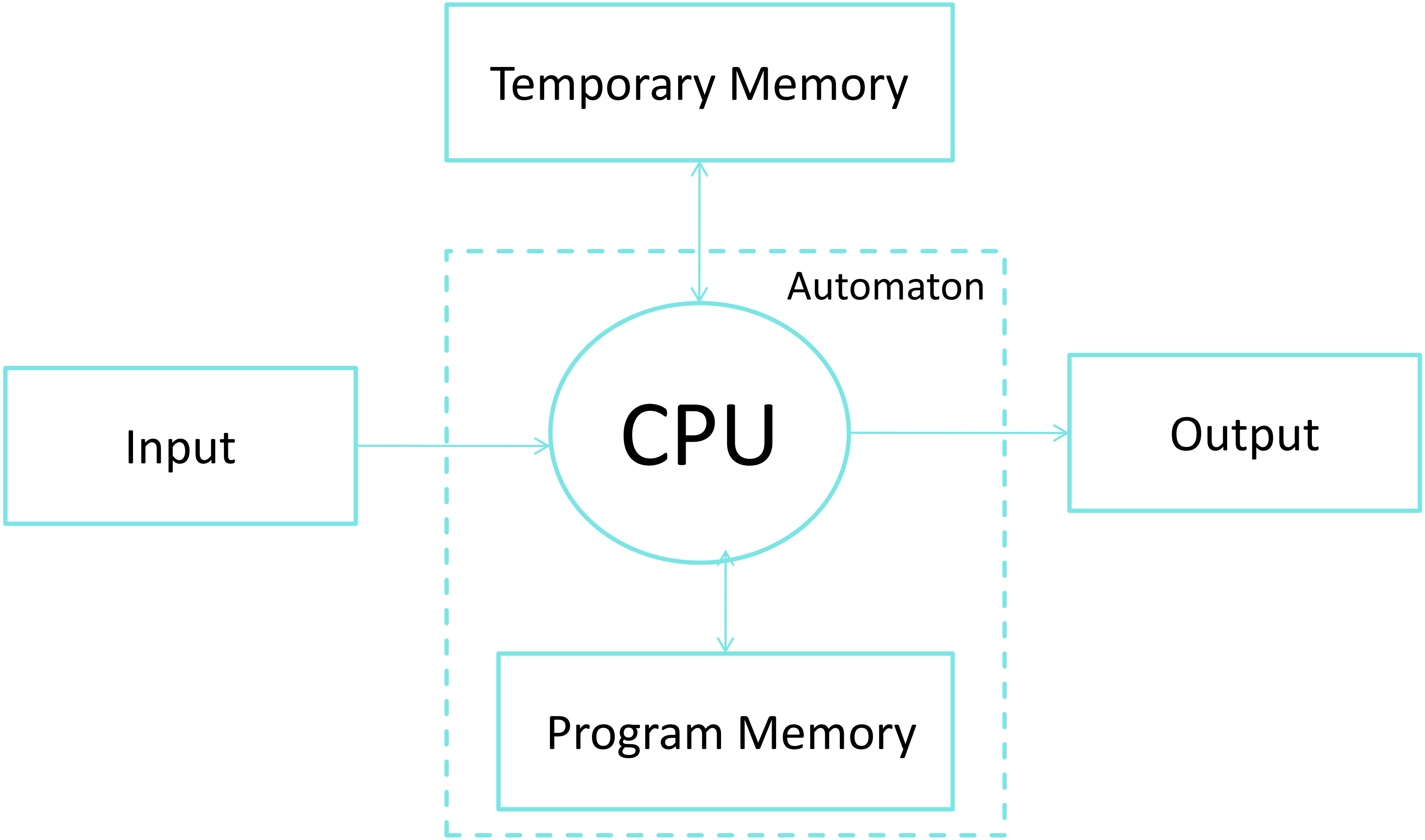
Topic mapping with Course Outcome

Topic	CO1	CO2	CO3	CO4	CO5
Finite Automata	3	-	-	-	-

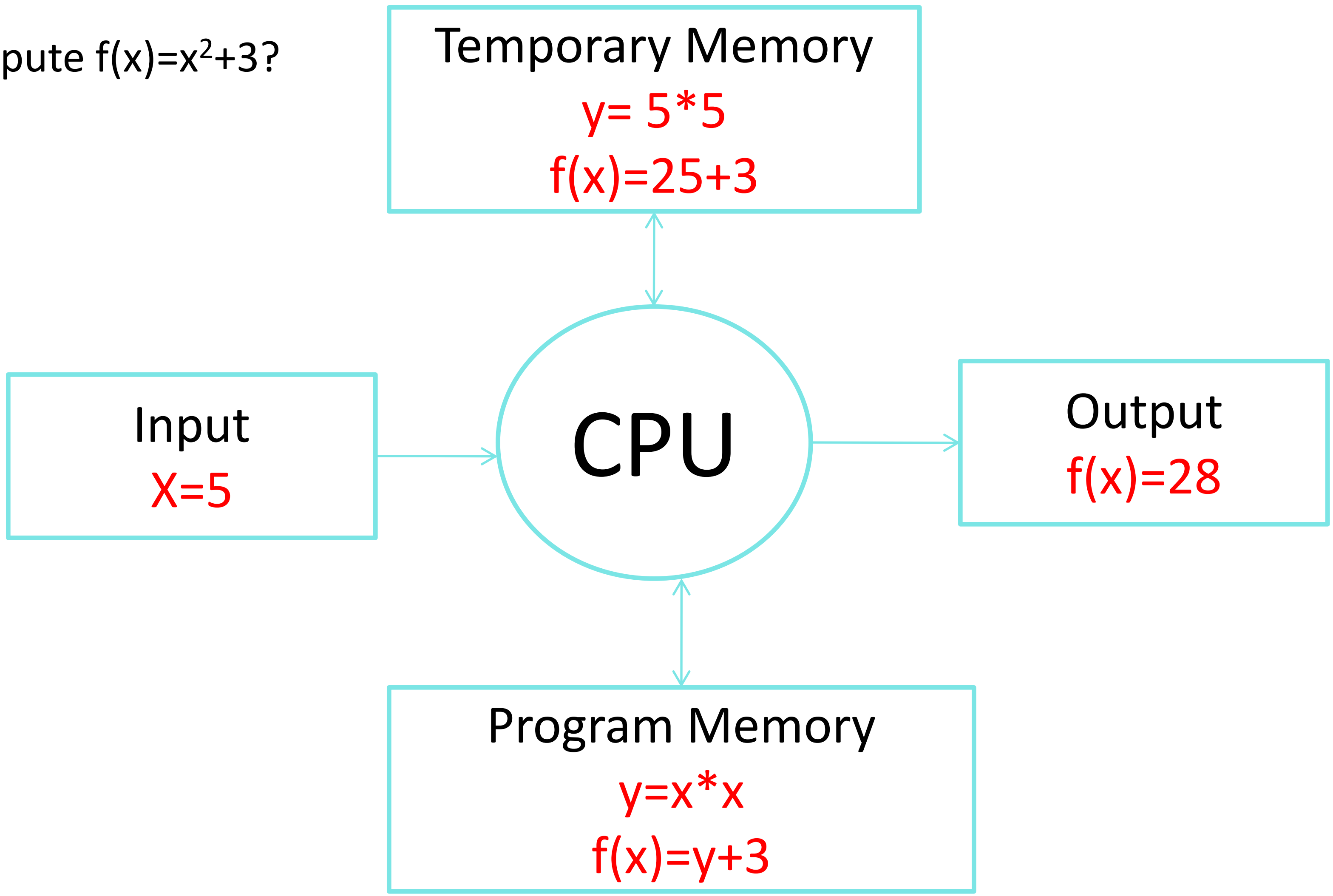
Introduction to Automata

- Automaton is the system that performs some function without human intervention.
- The plural of Automaton is Automata.

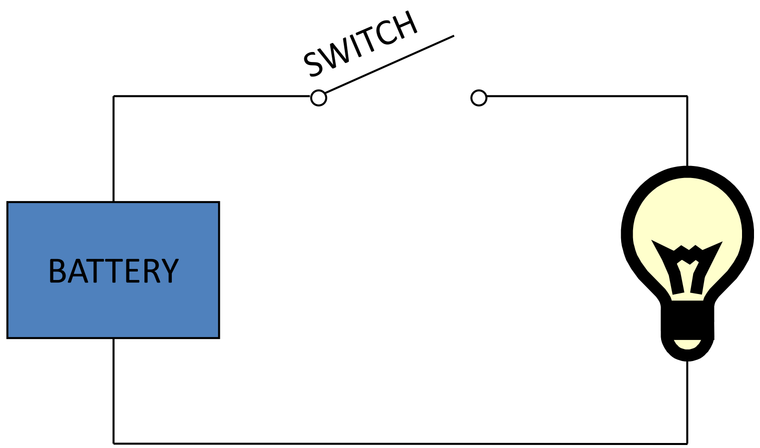
- Automata theory is the study of abstract computational machines and the computational problems that can be solved using these machines
- Abstract machine are (simplified) mathematical models of real computations


Temporary Memory
Automaton
Input
CPU
Output
Program Memory

Compute $f(x)=x^2+3$?



Why do we need abstract models?



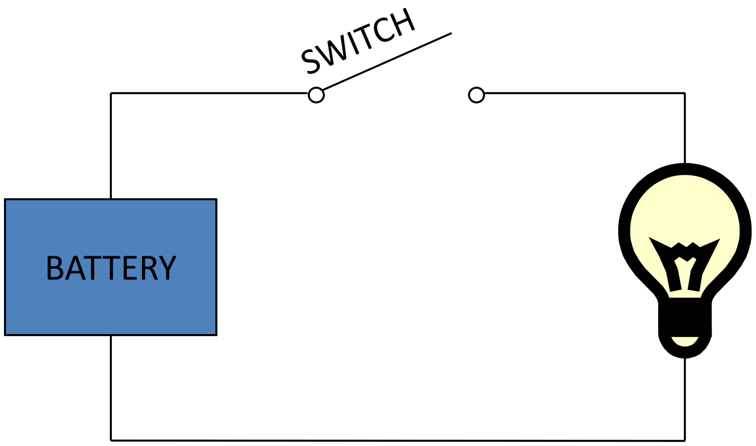
input: switch

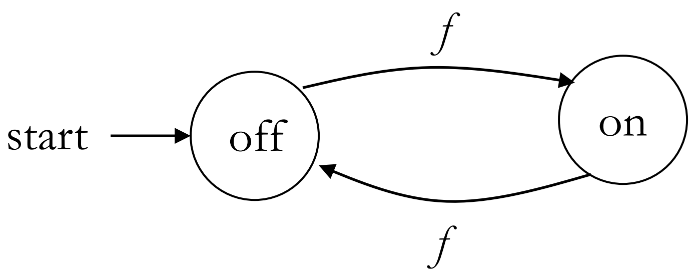
output: light bulb

actions: flip switch

states: on, off

A simple "computer"





input: switch

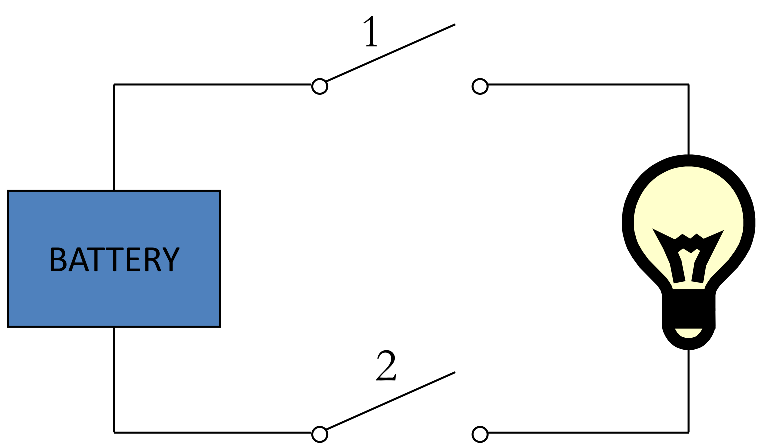
output: light bulb

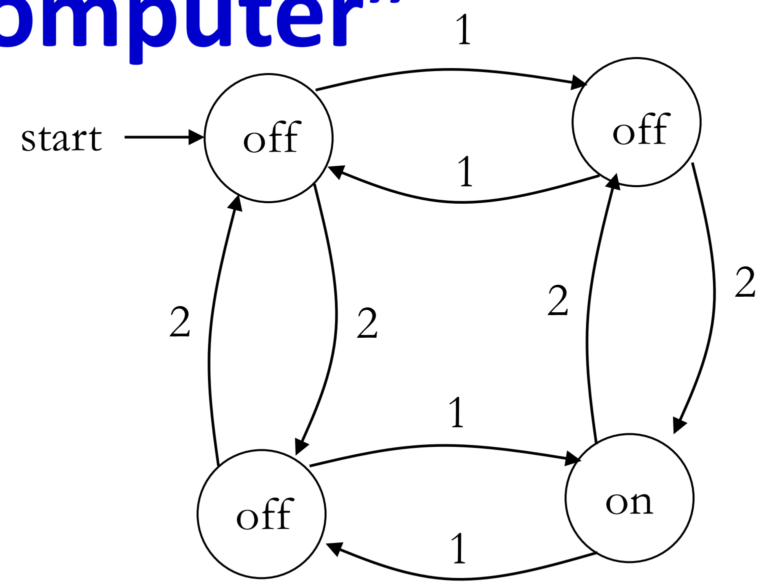
actions: *f* for "flip switch"

states: on, off

bulb is on if and only if there was an odd number of flips

Another "computer"





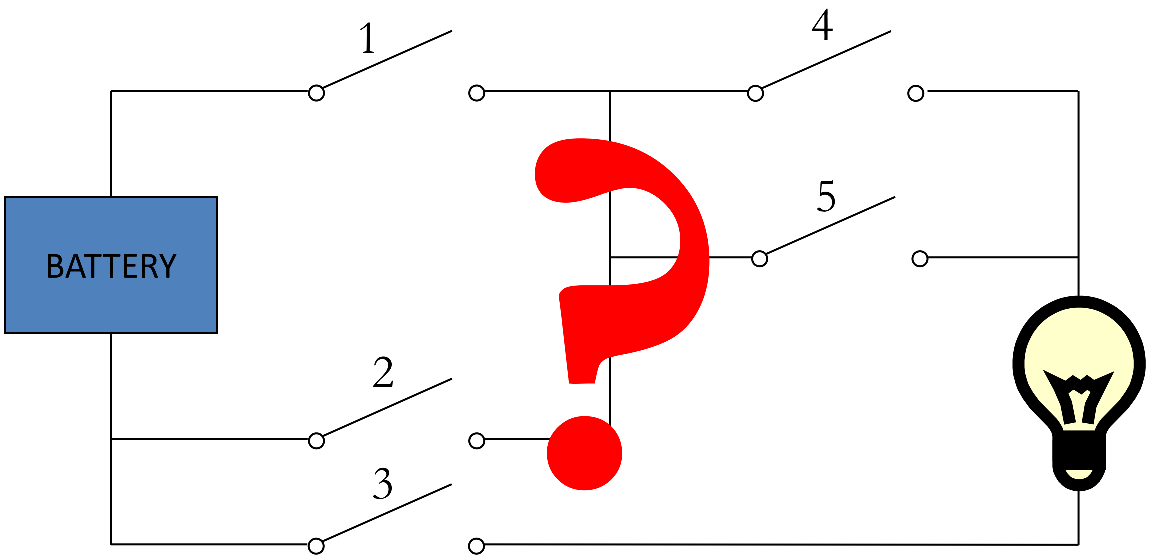
inputs: switches 1 and 2

actions: 1 for "flip switch 1"
2 for "flip switch 2"

states: on, off

bulb is on if and only if both switches were flipped an odd number of times

A Design Problem



Can you design a circuit where the light is on if and only if all the switches were flipped exactly the same number of times?

Such devices are difficult to reason about, because they can be designed in an infinite number of ways

A Design Problem

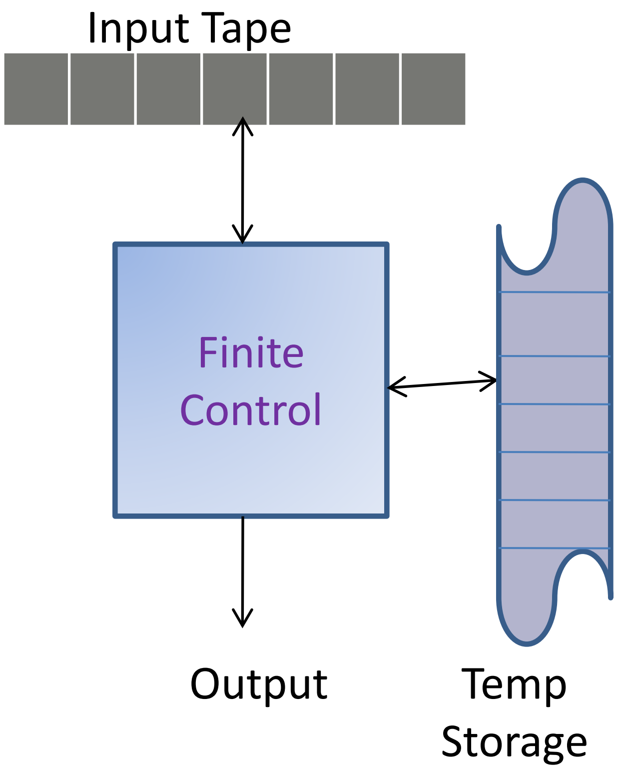
- By representing them as abstract computational devices, or automata, we will learn how to answer such questions

  - What can a given type of device compute, and what are its limitations?
  - Is one type of device more powerful than another?

Basic Concept

– Every automaton consists of some essential features as in real computers.

- ❑It has a mechanism for reading input. i.e. **Input Tape**
- ❑ The automaton can produce output of some form.
  - ❑If the output is Binary (accept or reject) - **Accepter.**
  - ❑Output sequence in response to an input sequence- **Transducer**
- ❑The automaton may have a temporary storage
- ❑The most important feature of the automaton is its **control unit.**



Different Types of Automata

finite automata	Devices with a finite amount of memory. Used to recognizer or string matching.
push-down automata	Devices with infinite memory that can be accessed in a restricted way. Used to model parsers, etc.
Turing Machines	Devices with infinite memory. Used to model any computer.
Linear Bounded Automata (time-bounded Turing Machines)	Infinite memory, but bounded running time. Used to model any computer program that runs in a "reasonable" amount of time.

Common terms used in Automata

Alphabet

An alphabet is a finite, non-empty set of symbols.

It is represented by $\sum$.

Ex:

$\sum=\{0,1\}$ is binary alphabet.

$\sum=\{0,1,2,3,4,5,6,7,8,9\}$ is decimal alphabet.

$\sum=\{a,b,c,\ldots\ldots,z\}$ is lower-case alphabet.

Common terms used in Automata

Strings/Words

A string or word is a finite sequence of symbols over ∑.

Ex:

w = 10110 is a string over ∑={0,1}.

Length of a string

or

Power of ∑

$\sum^k$ is the set of strings of length k ($|w|=k$).

Ex:

If $\sum=\{a,b\}$ then

$\sum^0 = \{\varepsilon\}$, called Epsilon.

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

$\sum^2 = \{aa,ab,ba,bb\}$

Common terms used in Automata

Reverse of a string

Reverse of a string, w is represented as w^R.

Ex:

If w=xyz then w^R =zyx.

Substrings of a string

If w=xyz then its substrings are:

Ɛ, x, y, z, xy, yz, zx, xyz

Common terms used in Automata

Prefix of a string

If w=xyz then its prefixes are:

Ɛ, x, xy, xyz

Suffix of a string

If w=xyz then its suffixes are:

Ɛ, z, yz, xyz

Languages

A set of strings, over ∑, is called a language.

Ex:

L_1=[w={0,1}* | w has equal number of 0's and 1's]

L_1={01,10,0011,0101,0110,1010,1001,1100,……}

L_2=[w={0,1}* | |w|$\leq$3]

L_2={Ɛ,0,1,00,01,10,11,000,001,010,011,100,101,110,111}

Kleene closure

Kleene closure represented as $\sum^*$, is set of strings of all possible length.

$$\sum^* = \sum^0 \cup \sum^1 \cup \sum^2 \cup \sum^3 \cup \ldots\ldots\ldots$$

Ex:

If $\sum = \{a,b\}$ then

$\sum^* = \{\varepsilon, a,b,aa,ab,ba,bb,aaa,aab,\ldots\ldots\ldots\}$

Positive closure

Positive closure represented as Σ^+, is set of strings of all possible length.

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots\ldots\ldots$$

$$\boxed{\Sigma^* = \Sigma^0 \cup \Sigma^+}$$

Ex:

If Σ ={a,b} then

Σ^+={a,b,aa,ab,ba,bb,aaa,aab,..........}

Grammar

A grammar **G** can be formally written as a 4-tuple (N, T, S, P) where −

- **N** or $\mathbf{V}_N$ is a set of variables or non-terminal symbols.
- **T** or ∑ is a set of Terminal symbols.
- **S** is a special variable called the Start symbol, $S \in N$
- **P** is Production rules for Terminals and Non-terminals. A production rule has the form $\alpha \rightarrow \beta$, where α and β are strings on $V_N \cup \Sigma$ and least one symbol of α belongs to V_N.

Example

Grammar G1 – ({S, A, B}, {a, b}, S, {S → AB, A → a, B → b})

Here,

- **S, A,** and **B** are Non-terminal symbols;
- **a** and **b** are Terminal symbols
- **S** is the Start symbol, S ∈ N
- Productions, **P : S → AB, A → a, B → b**

Grammar G2 –(({S, A}, {a, b}, S,{S → aAb, aA → aaAb, A → ε })

Here,

- **S** and **A** are Non-terminal symbols.
- **a** and **b** are Terminal symbols.
- **ε** is an empty string.
- **S** is the Start symbol, S ∈ N
- Production **P : S → aAb, aA → aaAb, A → ε**

Derivations from a Grammar

Strings may be derived from other strings using the productions in a grammar.

If a grammar **G** has a production **α → β**, we can say that **x α y** derives **x β y** in **G**.

This derivation is written as

$$x\,\alpha\,y \overset{G}{\Rightarrow} x\,\beta\,y$$

Example

Let us consider the grammar –

G2 = ({S, A}, {a, b}, S, {S → aAb, aA → aaAb, A → ε })

Some of the strings that can be derived are –

- S ⇒ aAb using production S → aAb
- ⇒ aaAbb using production aA → aaAb
- ⇒ aaaAbbb using production aA → aaAb
- ⇒ aaabbb using production A → ε

Language Generated by a Grammar

The set of all strings that can be derived from a grammar is said to be the language generated from that grammar.

A language generated by a grammar **G** is a subset formally defined by

$$L(G)=\{W \mid W \in \Sigma^*, S \Rightarrow_G \mathbf{W}\}$$

- If **L(G1) = L(G2)**, the Grammar **G1** is equivalent to the Grammar **G2**.

Example

1. If there is a grammar

   G: N = {S, A, B} T = {a, b} P = {S → AB, A → a, B → b}

- Here **S** produces **AB**, and we can replace **A** by **a**, and **B** by **b**. Here, the only accepted string is **ab**, i.e.,
- L(G) = {ab}

2. Suppose we have the following grammar –

   G: N = {S, A, B} T = {a, b} P = {S → AB, A → aA|a, B → bB|b}

- The language generated by this grammar –
- L(G) = {ab, a^2b, ab^2, a^2b^2, ………}

  = {a^m b^n | m ≥ 1 and n ≥ 1}

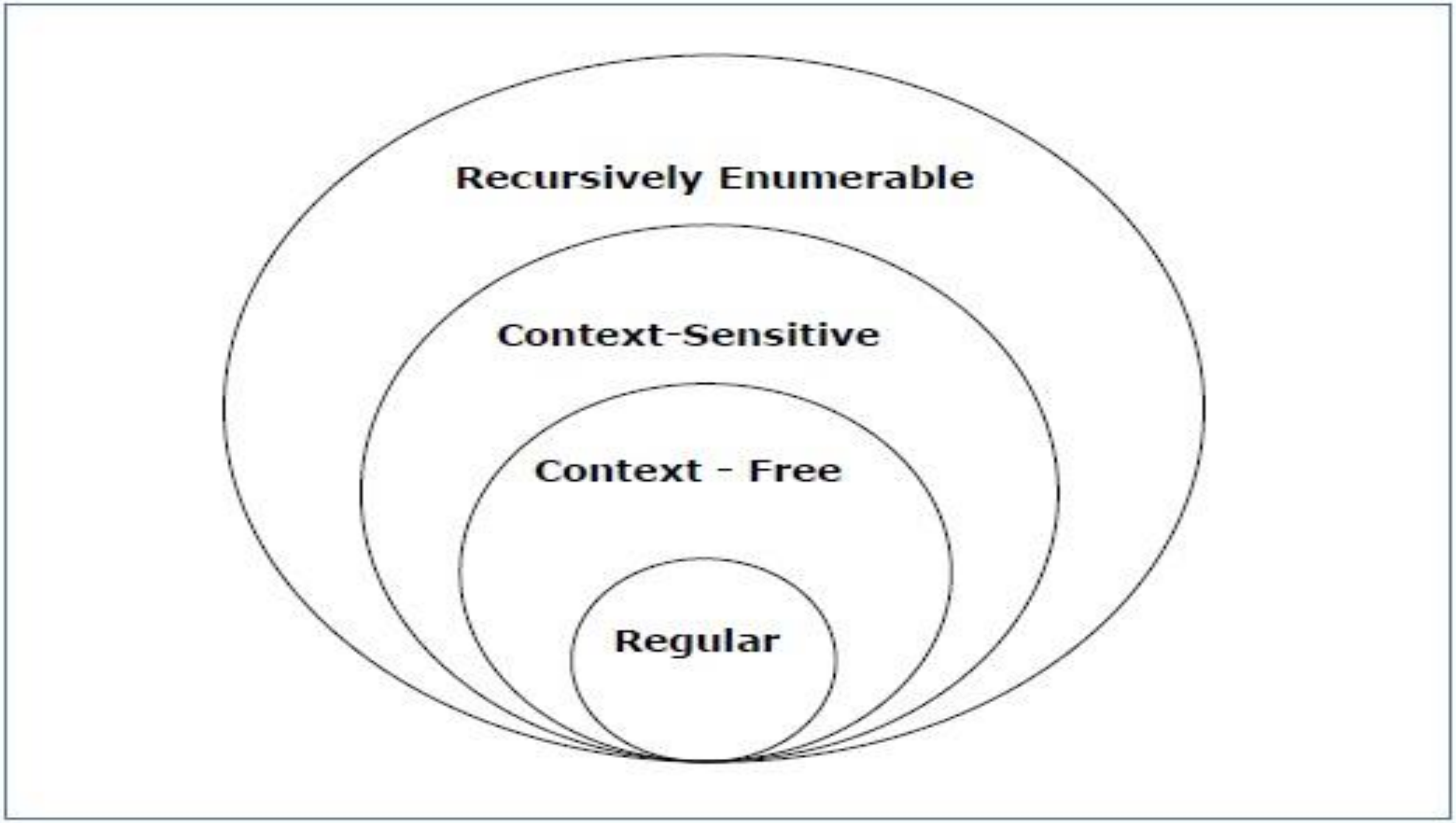
Chomsky Classification of Grammars

According to Noam Chomsky, there are four types of grammars – Type 0, Type 1, Type 2, and Type 3. The following table shows how they differ from each other

Grammar Type	Grammar Accepted	Language Accepted	Automaton
Type 0	Unrestricted grammar	Recursively enumerable language	Turing Machine
Type 1	Context-sensitive grammar	Context-sensitive language	Linear-bounded automaton
Type 2	Context-free grammar	Context-free language	Pushdown automaton
Type 3	Regular grammar	Regular language	Finite state automaton

Chomsky Classification of Grammars

It shows the scope of each type of grammar −



Type - 3 Grammar

- **Type-3 grammars** generate regular languages.
- Type-3 grammars must have a single non-terminal **on the left-hand side .**
- **a right-hand side** consisting of a single terminal or single terminal followed by a single non-terminal.
- The productions must be in the form

  X → a or X → aY ---------- (right linear)

  X → a or X → Ya -----------(left linear)

  where **X, Y ∈ N** (Non terminal)

  a ∈ T (Terminal)

Note: The rule **S → ε** is allowed if **S** does not appear on the right side of any rule.

Type - 3 Grammar Example

Example1 :

$X \rightarrow \varepsilon$

- $X \rightarrow a \mid aY$
- $Y \rightarrow b$

Example 2:

The language $0(10)^*$ is generated by

the right linear grammar: $S \rightarrow 0A$

$A \rightarrow 10\,A \,/\, \varepsilon$

The left linear grammar : $S \rightarrow S10 \,/\, 0$

Type - 2 Grammar

- **Type-2 grammars** generate context-free languages.
- The productions must be in the form **A → γ**

  where **A ∈ N** (Non terminal)

  γ ∈ (T ∪ N)* (String of terminals and non-terminals).

- These languages generated by these grammars are be recognized by a non-deterministic pushdown automaton.

Example:

S → X a

X → a

X → aX

X → abc

X → ε

NIET Greater Noida
GET FUTURE READY
AN AUTONOMOUS INSTITUTE

Type - 1 Grammar

- **Type-1 grammars** generate context-sensitive languages.
- The productions must be in the form

$$\alpha A \beta \rightarrow \alpha \gamma \beta$$

where **A ∈ N** (Non-terminal)

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

- The strings **α** and **β** may be empty, but **γ** must be non-empty.
- The rule **S → ε** is allowed if S does not appear on the right side of any rule. The languages generated by these grammars are recognized by a linear bounded automaton.
- **Right hand side string** be at least as long as or longer than **Left hand side string.**

Type - 1 Grammar Example

- $AB \rightarrow AbBc$
- $A \rightarrow bcA$
- $B \rightarrow b$

Type - 0 Grammar

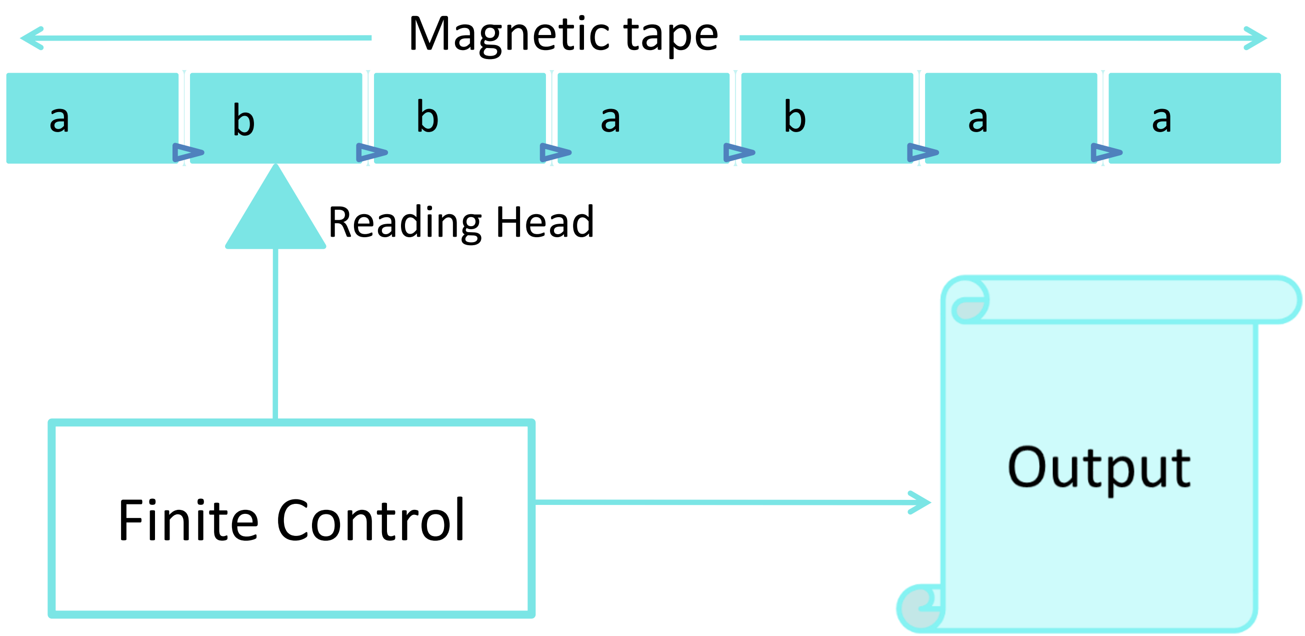
- **Type-0 grammars** generate recursively enumerable languages. The productions have no restrictions.
- They are any phase structure grammar including all formal grammars.
- They generate the languages that are recognized by a Turing machine.
- The productions can be in the form of **$\alpha \rightarrow \beta$**
- where **α** is a string of terminals and nonterminals with at least one non-terminal and **α** cannot be null. **β** is a string of terminals and non-terminals.

Type - 0 Grammar Example

- $S \rightarrow ACaB$
- $Bc \rightarrow acB$
- $CB \rightarrow DB$
- $aD \rightarrow Db$

Finite Automata

- Finite Automata: It is a mathematical model that works for several computer algorithms. It is called finite because it works on Finite set of Input symbols with Finite number of States and gives output in finite times.



Applications of Automata

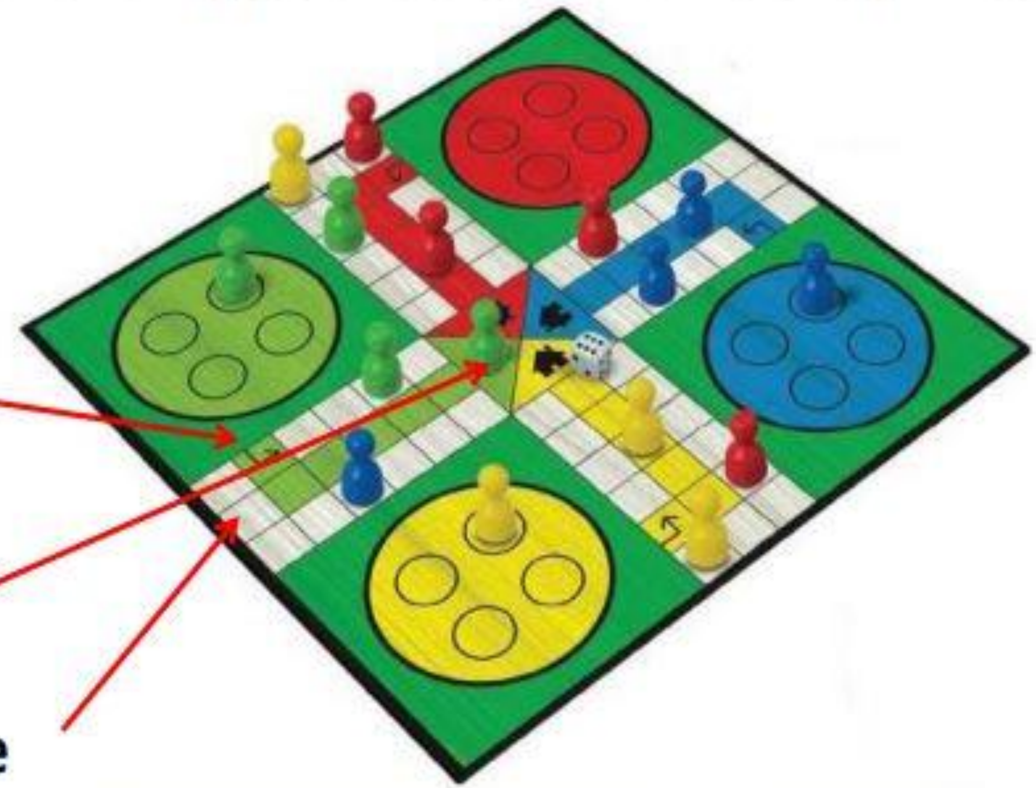
Applications of Finite Automata

- String Matching
- Lexical Analysis
- Design and Analysis of Digital Circuits

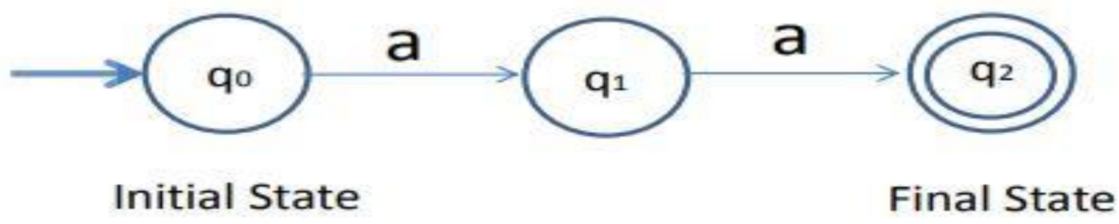
- ➢ Finite automata is another method for defining languages.
- ➢ It's a Graphical Method.

Token changes Position on the input of certain number by Dice

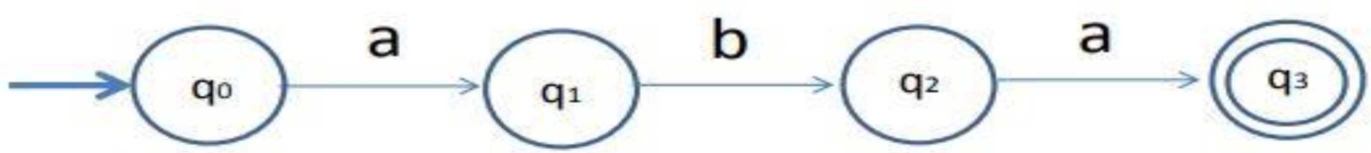
State



L1 = {aa}



L2 = {aba}



Why is it named as Finite Automata?

Automata : Plural of Automaton (i.e. automatic , self controlled machine)

Finite : Finite number of states / letters / transitions.

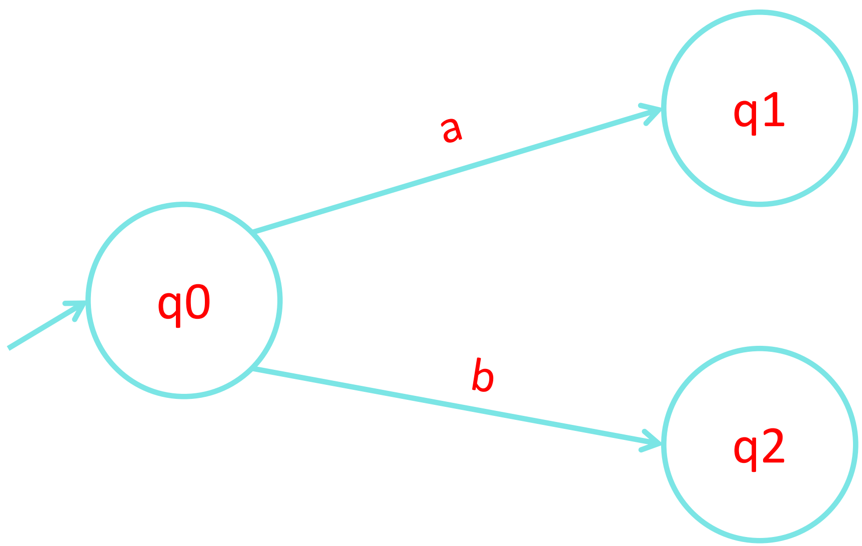
Finite Automata

Finite Automata

- Deterministic Finite Automata (DFA)
- Non-deterministic Finite Automata (NFA)

DFA (Deterministic Finite Automata)

- For a DFA the machine moves to a particular unique state from a given state and given input that is why it is called Deterministic. As the number of states in DFA are Finite it is called Deterministic Finite Automata.
- Next State is known (Has only one next state for an input)



Deterministic Finite Automata

- A deterministic finite automaton (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where
  - Q is a finite set of states
  - Σ is finite set of alphabet
  - $\delta: Q \times \Sigma \rightarrow Q$ is a transition function
  - $q_0 \in Q$ is the initial state
  - $F \subseteq Q$ is a set of accepting states (or final states).
- In diagrams, the accepting states will be denoted by double loops

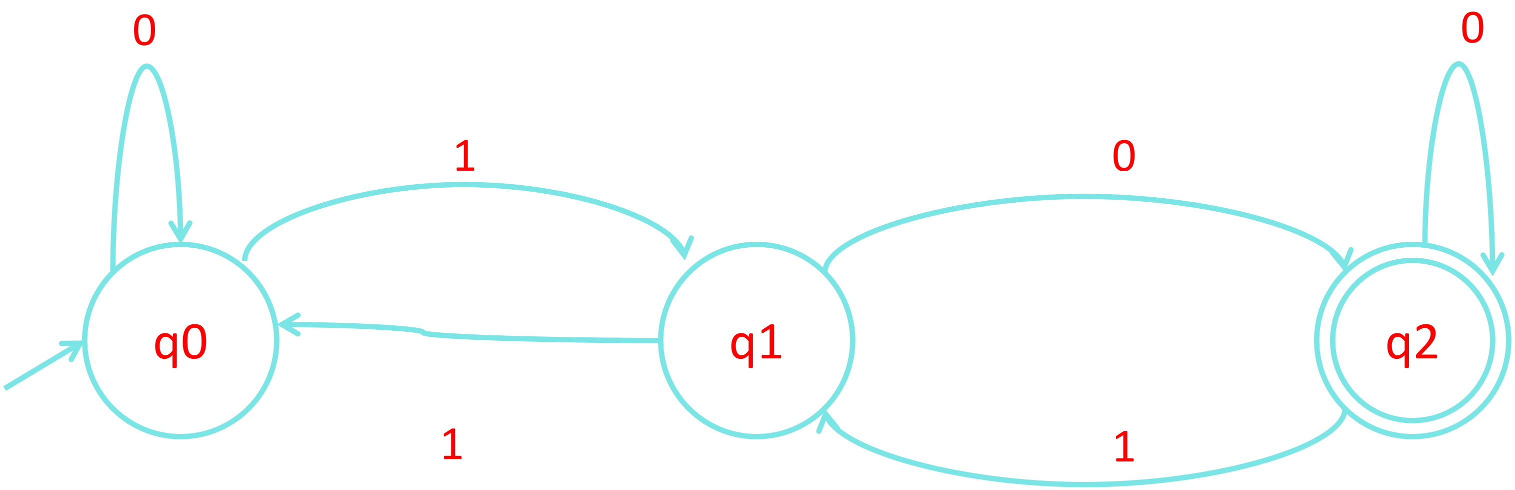
DFA

Representation of δ

- Transition Diagram
- Transition Table
- Transition Function

DFA

Transition Diagram



M={ {q0, q1, q2}, {0,1}, δ, q0, q2}

DFA

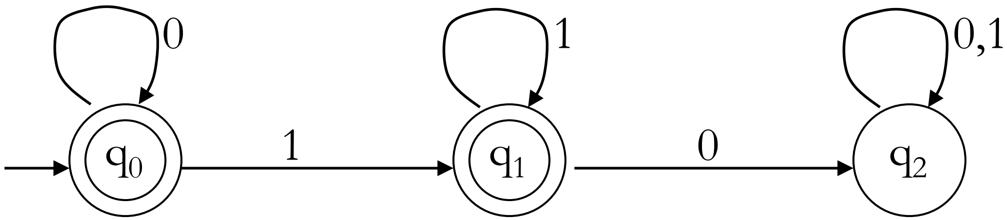
Transition Table

Present State	Next State	
	0	1
$\rightarrow$ q0	q0	q1
q1	q0	q2
*q2	q0	q0

Transition Function

$$\delta(q_0, 0) = q_0$$
$$\delta(q_0, 1) = q_1$$
$$\delta(q_1, 0) = q_2$$
$$\delta(q_1, 1) = q_0$$
$$\delta(q_2, 0) = q_1$$
$$\delta(q_2, 1) = q_2$$

Example



alphabet $\Sigma = \{0, 1\}$
start state $Q = \{q_0, q_1, q_2\}$
initial state q_0
accepting states $F = \{q_0, q_1\}$

transition function δ:

inputs

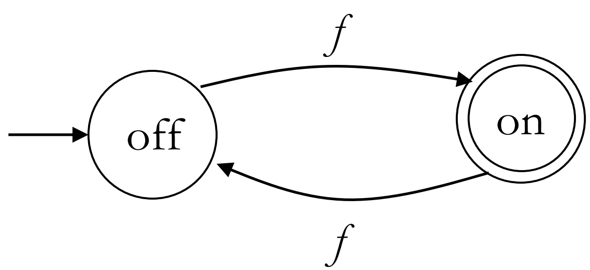
states	0	1
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_2	q_2

Language of a DFA

The language of a DFA M=$(Q, \Sigma, \delta, q_0, F)$ is the set of all strings over Σ accepted by M *i.e.* L(M) and is denoted

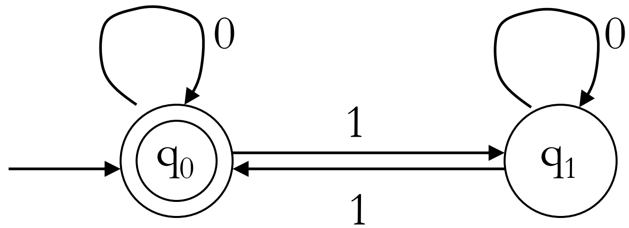
$$L(M) = \{w \mid w \in \Sigma^* , \delta(q_0 , w) = q_f \text{ where } q_f \in F\}$$

M:

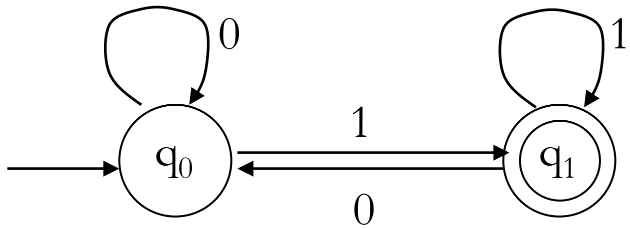


- Language of M is $\{f, fff, fffff, \ldots\} = \{f^n : n \text{ is odd}\}$

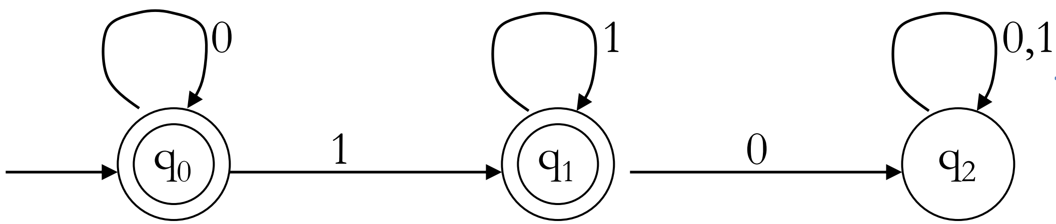
Example



The set of all string over {0,1} containing even no. of 1's



The set of all string over {0,1} ending with 1's



The set of all string over {0,1} does not contain 10 as substring

Variation of Finite Automata

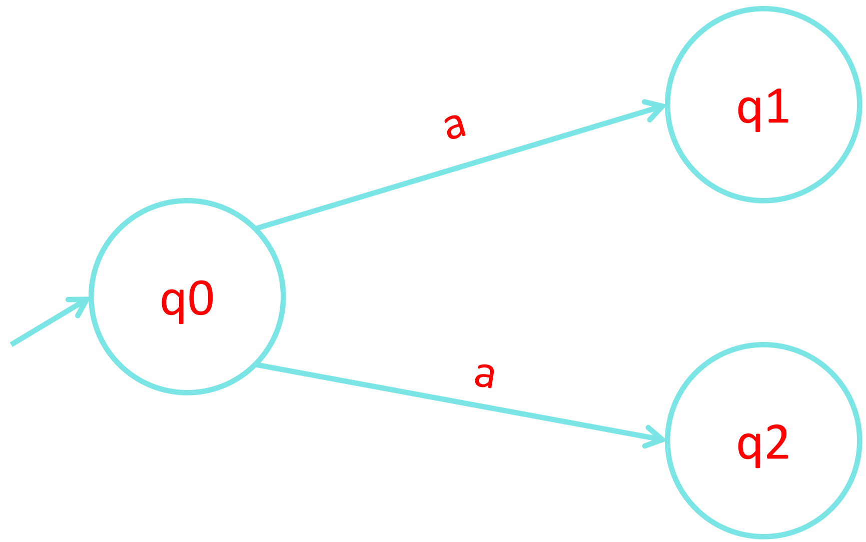
- Deterministic Finite Automat
  - $\delta: Q \times \Sigma \rightarrow Q$
- Non-Deterministic Finite Automata
  - $\delta: Q \times \Sigma \rightarrow 2^Q$
- Non-Deterministic Finite Automata with Epsilon
  - $\delta: Q \times \{\Sigma \cup \{\varepsilon\}\} \rightarrow 2^Q$
- Finite Automata with output

  $(Q, \Sigma, \Delta, \delta, \lambda, q0)$ where Q, Σ, δ, and $q0$ are similar to DFA and Δ is the set of output symbol and λ is out put function

  - **Moore Machine** : the output is depend on present state $\lambda : Q \rightarrow \Delta$
  - **Mealy Machine** : the output is depend on present state and current input applied on the state $\lambda : Q \times \Sigma \rightarrow \Delta$

NFA

In NFA, the machine can move from a present state to a combination of states for an input symbol that is why it is called non-deterministic. As the number of states are finite, it is called Non-deterministic Finite Automata.



Formal definition of an NFA

- An NFA is represented as a 5 tuples,

  $M=\{Q, \Sigma, \delta, q_0, F\}$

  Q= Non empty finite set of states

  Σ= Input alphabet

  δ= Transition Function, $\delta: Q \times \Sigma \rightarrow 2^Q$

  q_0= Initial State

  F= Set of final states ($F \subseteq Q$)

Difference between DFA and NFA

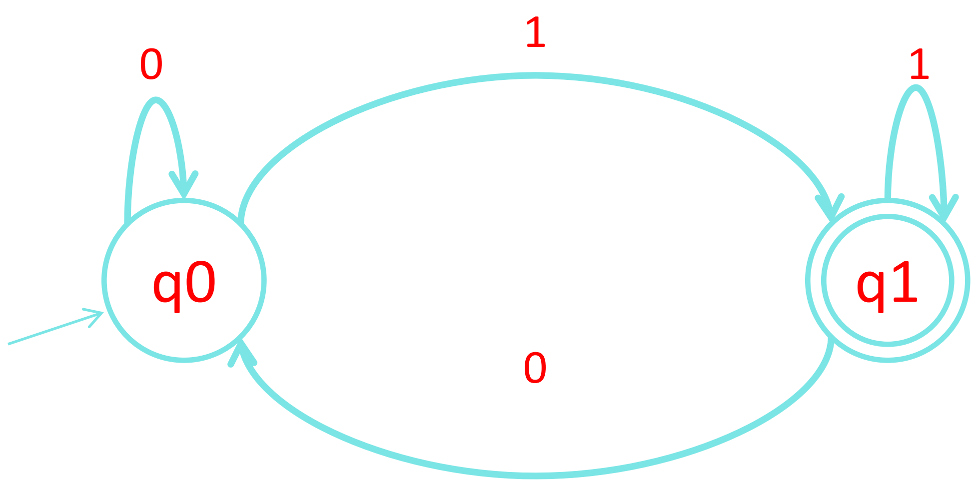
The transition from a state is to a single particular next state for each input symbol. Hence it is called *deterministic*.	The transition from a state can be to a combination of states for each input symbol. Hence it is called *non-deterministic*.
Ɛ-transition is not allowed in DFA.	NFA permits Ɛ-transition.
DFA takes more space.	NFA takes less space.
DFA is as powerful as NFA.	NFA is as powerful as DFA.
A string is accepted by a DFA, if it transits to a final state.	A string is accepted by a NDFA, if at least one of all possible transitions ends in a final state.
Every DFA is an NFA.	Every NFA is not a DFA.

Examples of DFA

- Construct a DFA that accepts all strings over ∑={0,1} ending with 1.
- Construct a DFA for L={w=(0,1)* | |w|=3n , n=0,1,2,...}
- Construct a DFA for L= { all binary strings containing substring 001 }

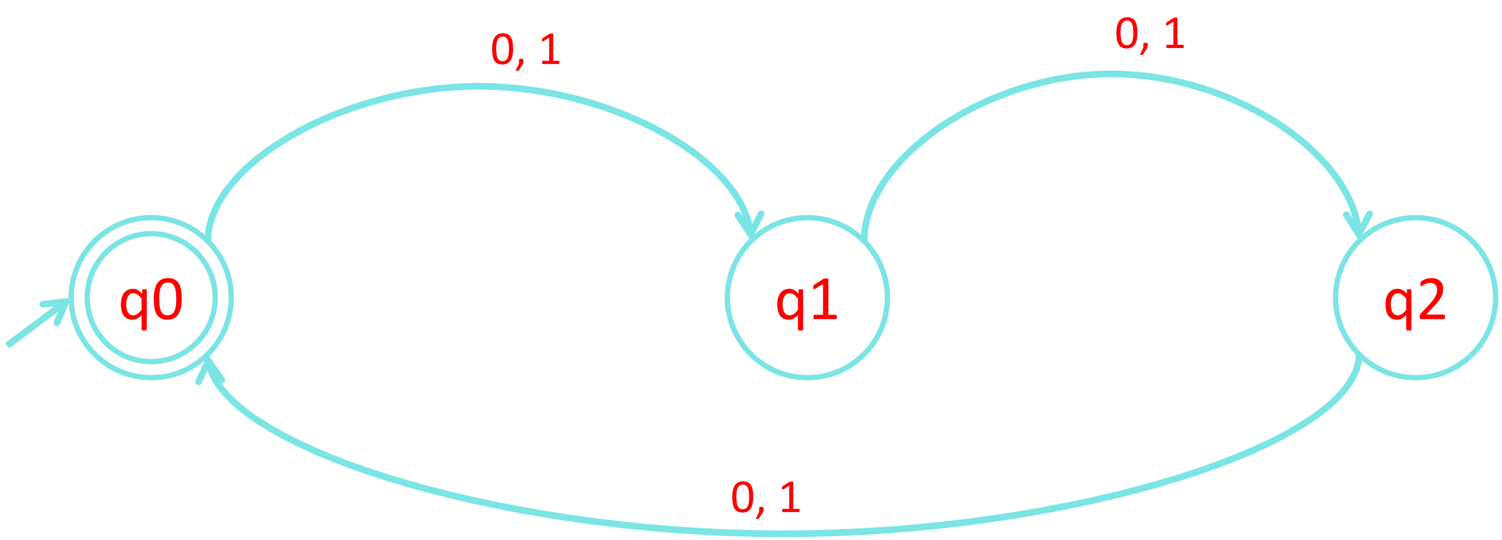
Examples of DFA

- Construct a DFA that accepts all strings over ∑={0,1} ending with 1.



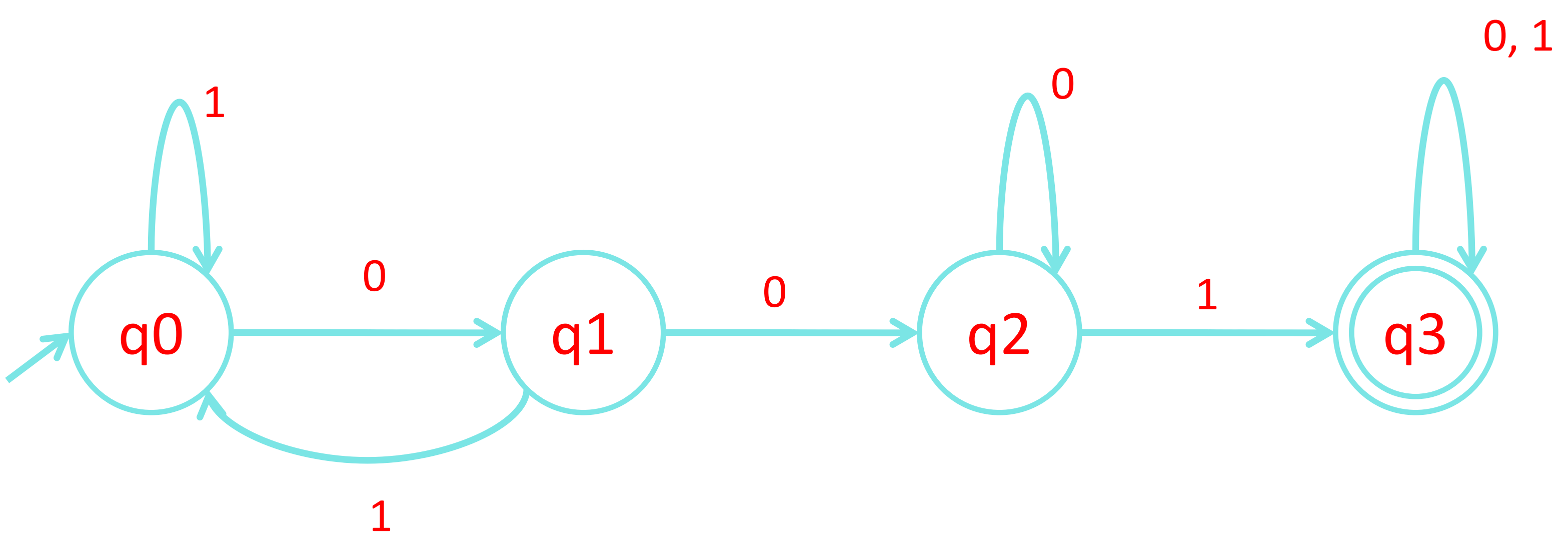
Examples of DFA

- Construct a DFA for L={w=(0,1)* | |w|=3n , n=0,1,2,...}



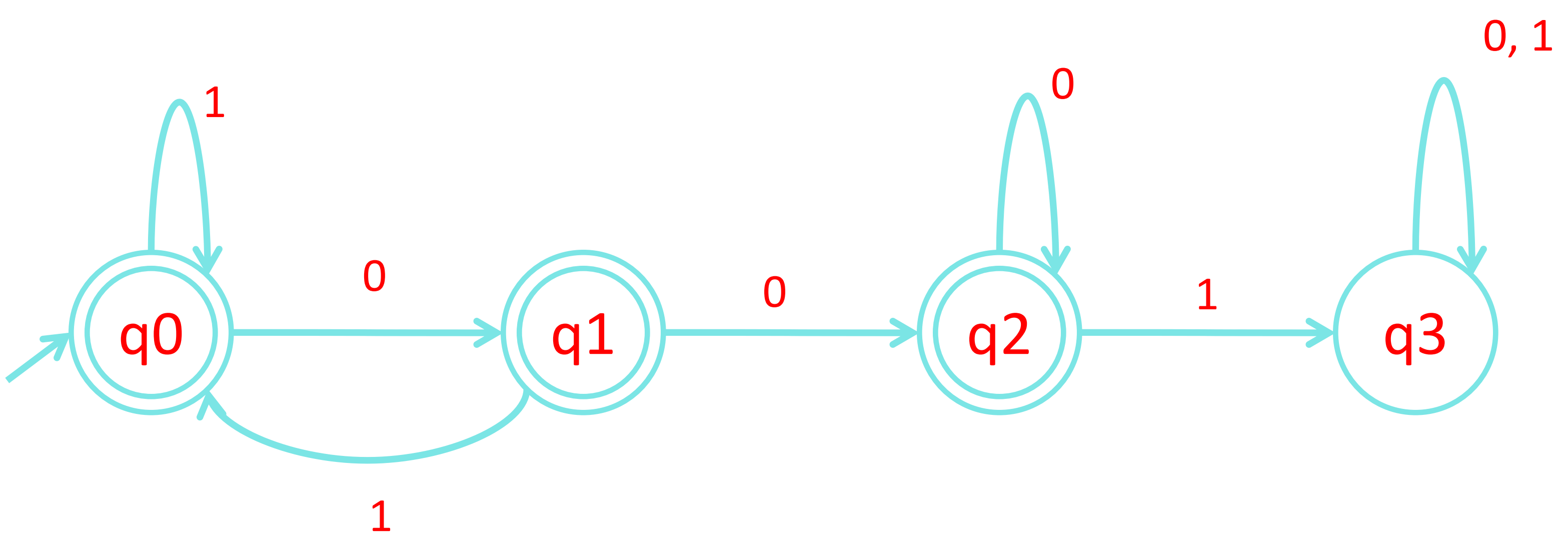
Examples of DFA

- Construct a DFA for L= { all binary strings containing substring 001 }



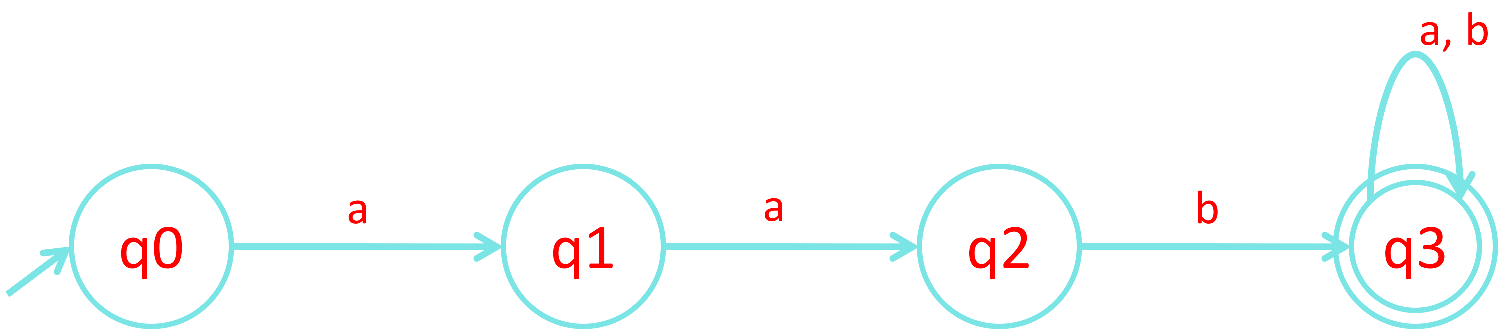
Examples of DFA

- Construct a DFA for L= { all binary strings not containing substring 001 }



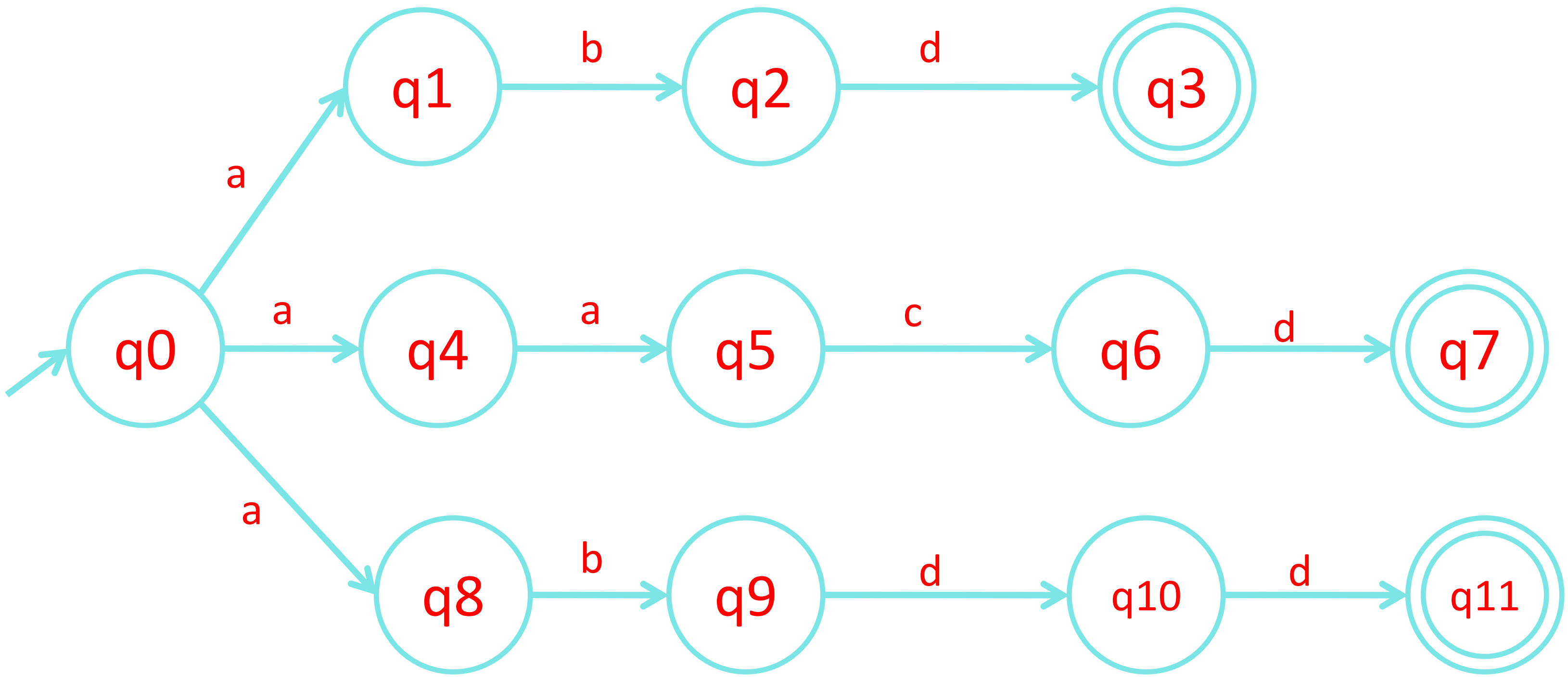
Examples of NFA

- Construct a NFA for L= { all strings beginning with aab }

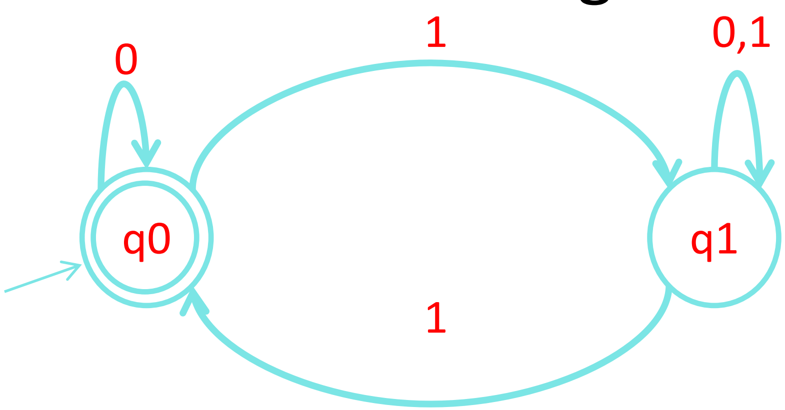


Examples of NFA

- Construct a NFA over ∑={a,b,c,d} that recognises abd, aacd or abdd.

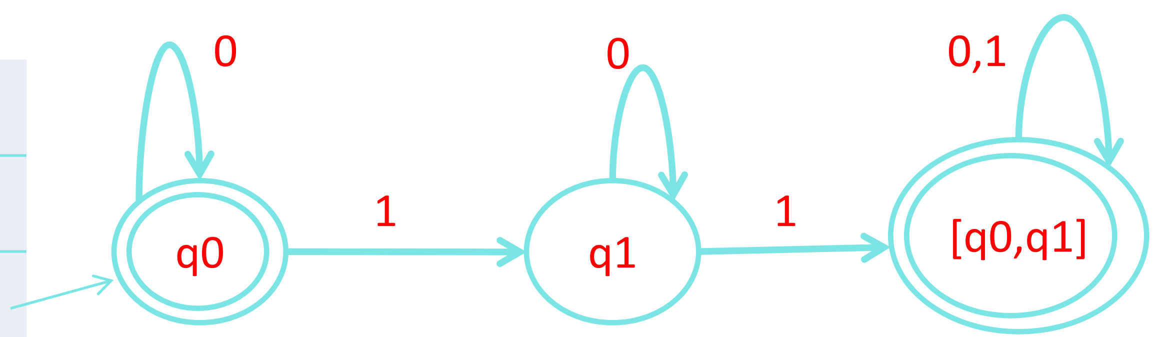


- Convert following NFA to DFA



	0	1
q0	q0	q1
q1	q1	{q0,q1}

	0	1
q0	q0	q1
q1	q1	[q0,q1]
[q0,q1]	[q0,q1]	[q0,q1]



Theorem: Every NFA has an equivalent DFA.

If a language L is accepted by an NFA then there exists an equivalent DFA that accepts L.

Equivalence of NFA and DFA (Proof)

Let L is accepted by NFA $N = (Q_N, \Sigma_N, \delta_N, q_{0N}, F_N)$.

Construct a DFA $D= (Q_D, \Sigma_D, \delta_D, q_{0D}, F_D)$ as follows.

Q_D is equal to the power set of Q_N, $Q_D = 2^{Q_N}$

$\Sigma_D = \Sigma_N = \Sigma$

$q_{0D} = \{q_{0N}\}$

F_D is the set of states in Q_D that contain any element of F_N.

δ_D is the transition function for D.

$\delta_D (q,a)= \cup_{p \in q} \delta_N (q,a)$ for $q \in Q_D$ and $a \in \Sigma$.

p is a single state from Q_N .

$\delta_D(q,a)$ is the union of all $\delta_N(p,a)$.

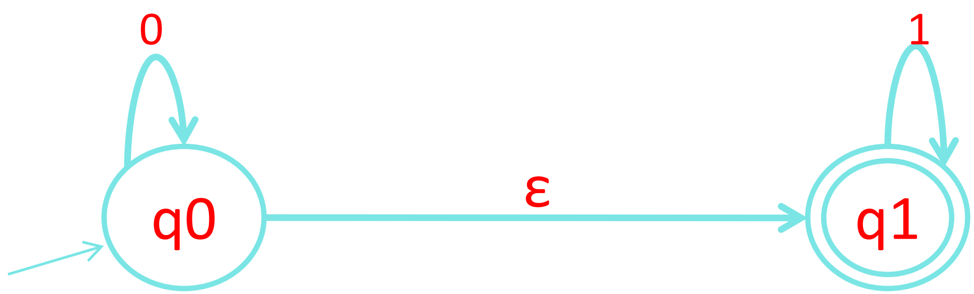
Now we will prove that for every *x*, L(D) = L(N)

Basis Step

Let *x* be the empty string ε.

$$\begin{aligned}\delta_D(q_{0D}, x) &= \delta_D(q_{0D}, \varepsilon)\\ &= q_{0D}\\ &= \{q_{0N}\}\\ &= \delta_N(q_{0N}, \varepsilon)\\ &= \delta_N(q_{0N}, x)\end{aligned}$$

NFA with Ɛ- Transition



We extend the class of an NFA by allowing ε transitions:

- The automaton may be allowed to change its state without reading the input symbol.
- Such transitions are depicted by labeling the appropriate arcs with ε.
- ‘ε’ does not belong to any alphabet (∑).

Why Ɛ- NFA ?/

- ε -NFAs add a convenient feature.
- Through ε –NFAs we can implement some complex languages easily.
- They do not extend the power of an NFA.
- Both NFAs and ε-NFAs have same power.

Formal definition of an Ɛ- NFA

- An NFA is represented as a 5 tuples,

  $M=\{Q, \Sigma, \delta, q_0, F\}$

  Q= Non empty finite set of states

  ∑= Input alphabet

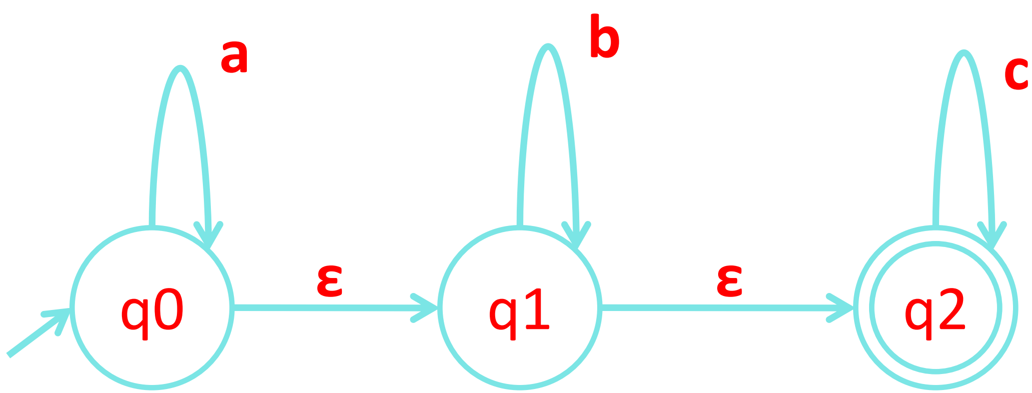
  δ= Transition Function, $\delta: Q \times (\Sigma \cup \varepsilon) \rightarrow 2^Q$

  q_0= Initial State

  F= Set of final states ($F \subseteq Q$)

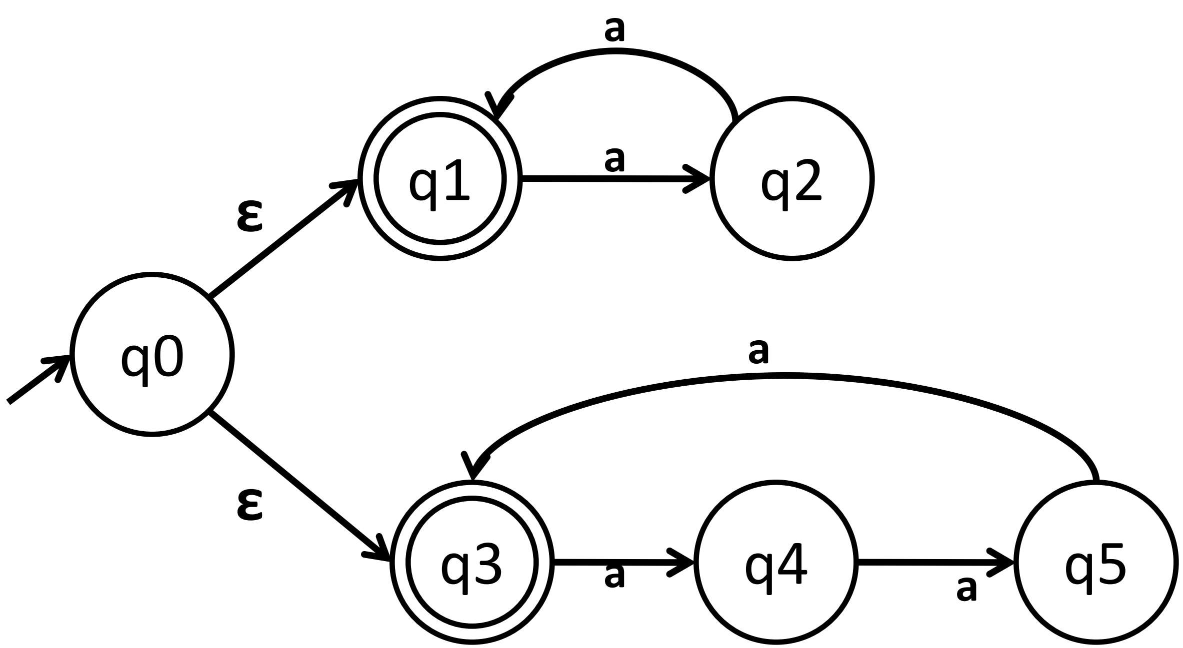
Example of ε -NFA

- Design an NFA for L={ $a^p b^q c^r$ | p,q,r ≥0}.



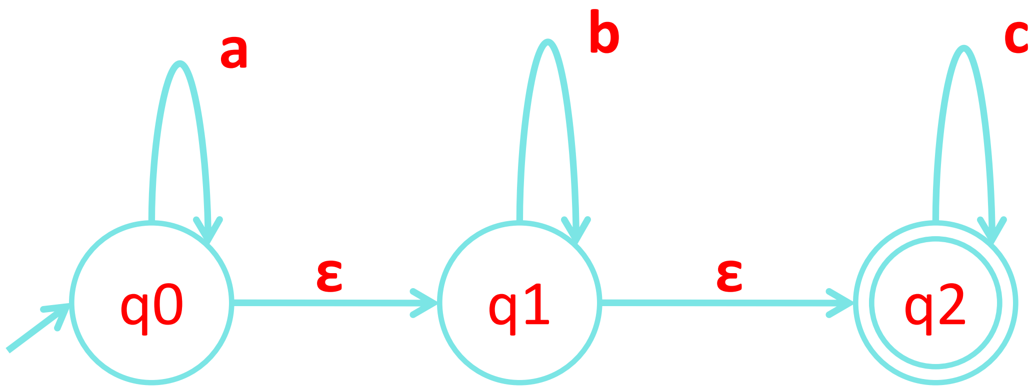
Example of ε -NFA

- Design an NFA for L={ a^n| n is even or divisible by 3}.



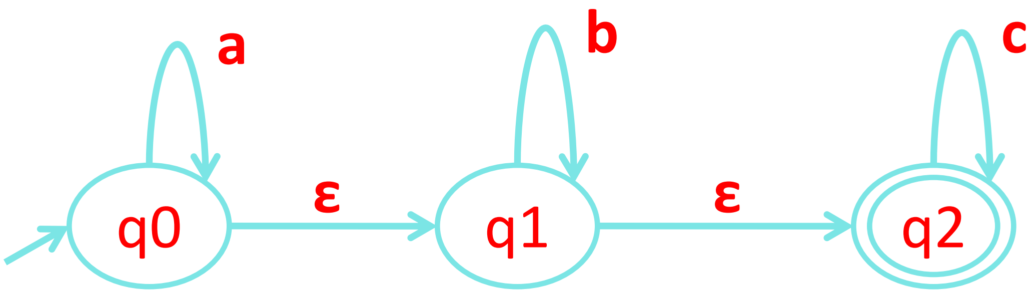
ε –Closure of a state

- The ε-closure of the state q, denoted as ε-Closure(q), is the set that contains q, together with all states that can be reached starting at q by following only ε-transitions.



- ε-Closure(q0) = {q0, q1, q2}
- ε-Closure(q1) = {q1, q2}
- ε-Closure(q2) = {q2}

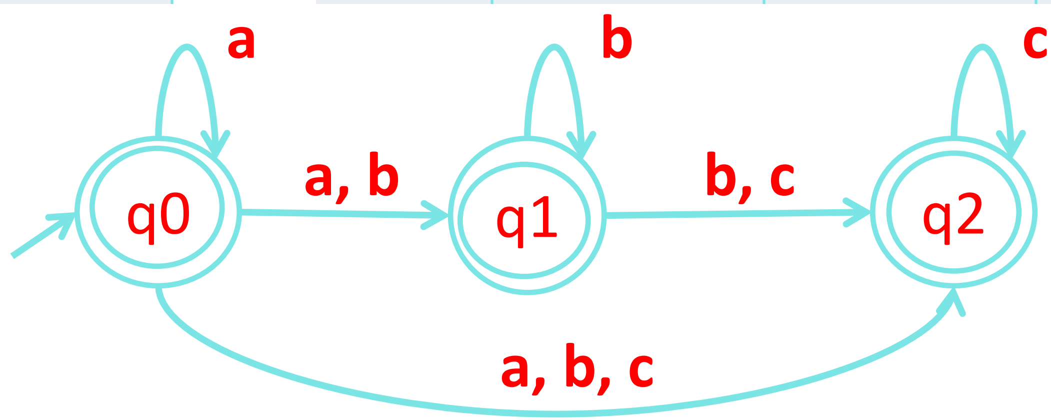
Conversion of ε –NFA to NFA



δ(q0, a)= ε-Closureδ(ε-Closure(q0), a)
= ε-Closureδ({ q0,q1,q2},a)
= ε-Closure(q0)
= {q0, q1, q2}

States	ε-Closure
q0	{q0, q1, q2}
q1	{q1, q2}
q2	{q2}

	a	b	c
q0	{q0, q1, q2}	{q1, q2}	{q2}
q1	ϕ	{q1, q2}	{q2}
q2	ϕ	ϕ	{q2}



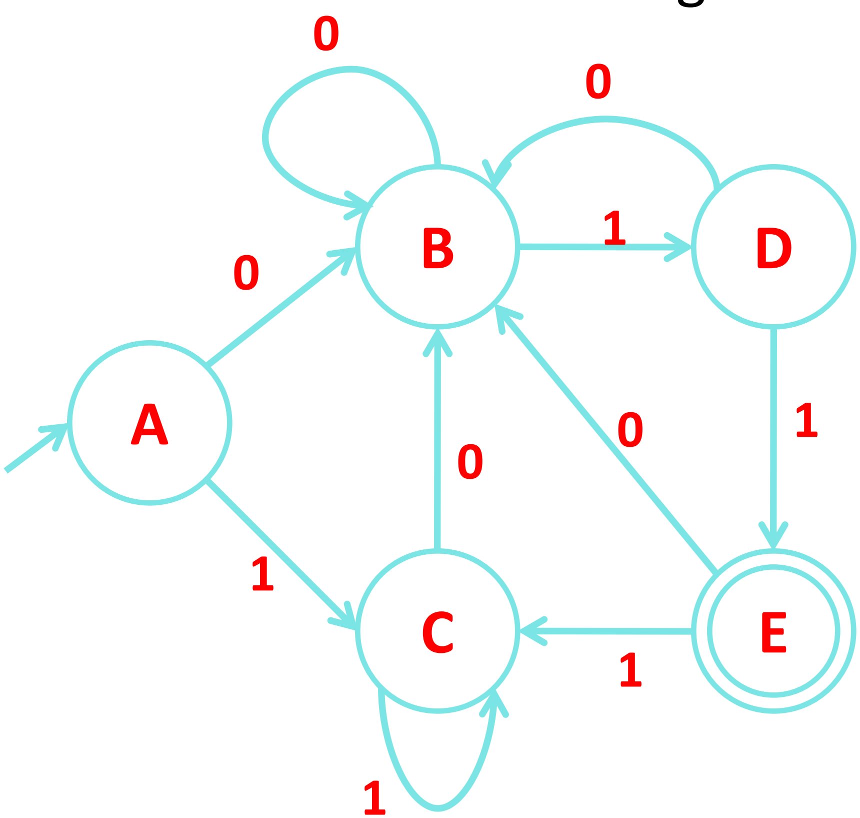
Minimization of DFA

- Minimization of a DFA refers to the removal of those states of a DFA, whose presence or absence in a DFA does not affect the language accepted by the automata.

- The states that can be eliminated from automata are:
  - ➢ Unreachable or inaccessible states.
  - ➢ Dead states.
  - ➢ Non-distinguishable or indistinguishable state or equivalent states.

Minimization of DFA

- Minimize the following DFA:



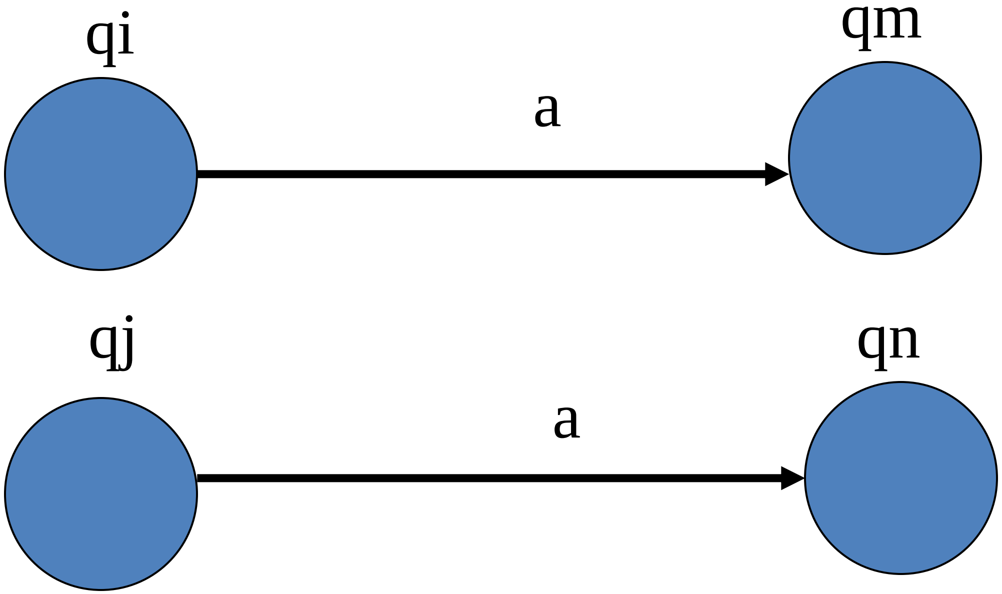
Transition Table

	0	1
→A	B	C
B	B	D
C	B	C
D	B	E
*E	B	C

Minimization of DFA

- Eqivalent State

Two states qi and qj are *equivalent* or *indistinguishable* (for the future), if, when started in these states, every string causes the machine to either end up in an accepting state for both or end up in a non-accepting state for both.



If qm and qn are distinguishable, then so are qi and qj

Minimization of DFA

Suppose there is a DFA D < Q, Σ, q0, δ, F > which recognizes a language L. Then the minimized DFA D < Q', Σ, q0, δ', F' > can be constructed for language L as:

- **Step 1:** We will divide Q (set of states) into two sets. One set will contain all final states and other set will contain non-final states. This partition is called Π_0.
- **Step 2:** Initialize k = 1
- **Step 3:** Find Πk by partitioning the different sets of Πk-1. In each set of Πk-1, we will take all possible pair of states. If two states of a set are distinguishable, we will split the sets into different sets in Πk.
- **Step 4:** Stop when $\Pi k = \Pi k$-1 (No change in partition)
- **Step 5:** All states of one set are merged into one. No. of states in minimized DFA will be equal to no. of sets in Πk.

Minimization of DFA

Transition Table

	0	1
→A	B	C
B	B	D
C	B	C
D	B	E
*E	B	C

Find Equivalence classes

Π_0= {A,B,C,D} {E}

Π_1= {A,B,C} {D} {E}

Π_2= {A,C} {B} {D} {E}

Π_3= {A,C} {B} {D} {E}

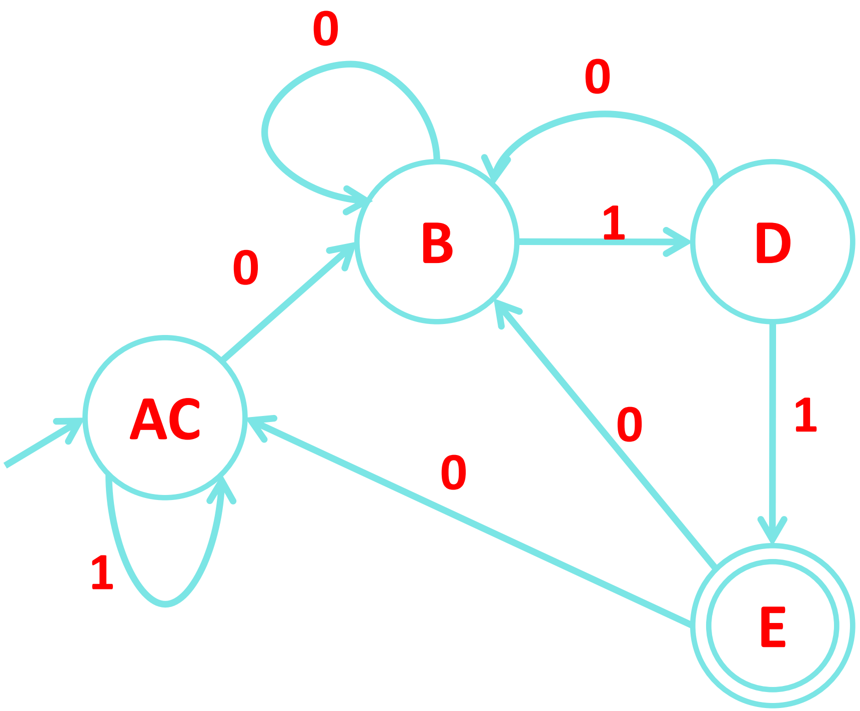
$\Pi_2 = \Pi_3$,so stop here.

Minimization of DFA

{A,C} {B} {D} {E}

Transition Table

	0	1
→A	B	C
B	B	D
C	B	C
D	B	E
*E	B	C



Using Table Filling Method (Myhill-Nerode Theorem)

Step 1: Draw a table for all pairs of states (P,Q) not necessarily connected directly [All are unmarked initially].

Step 2: Consider every state pair (P,Q) in the DFA where P ∈ F and Q ∉ F or vice versa and mark (X) them. [Here F is the set of final states].

Using Table Filling Method (Myhill-Nerode Theorem)

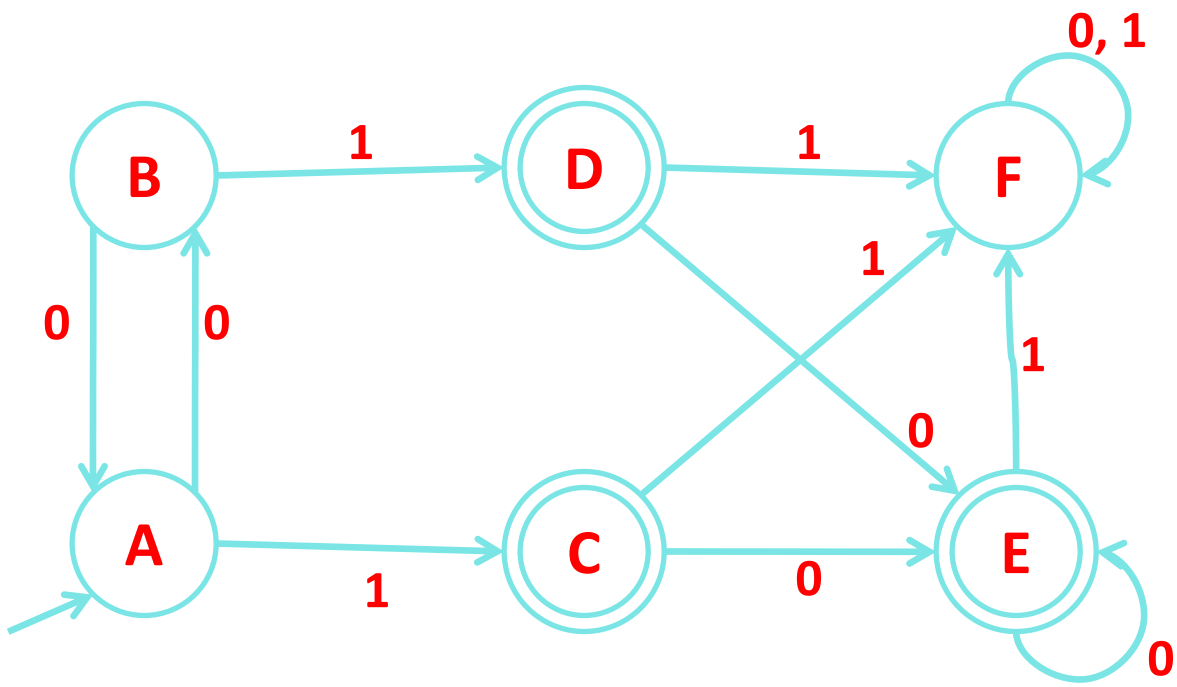
Contd...

Step 3: Repeat this step until we cannot mark anymore states – If there is an unmarked pair (P, Q), such that {δ(P , a), δ (Q , a)} is marked then mark (P,Q), where a ∈ ∑.

Step 4: Combine all the unmarked pair (P , Q) and make them a single state in the reduced DFA.

Minimization of DFA

Minimize the following DFA.



Minimization of DFA

Step 1: Draw a Table for all pair of states (P,Q).

	A	B	C	D	E	F
A						
B						
C						
D						
E						
F						

Minimization of DFA

Step 2: Mark every state pair (P,Q) in the DFA where P ∈ F and Q ∉ F .

	A	B	C	D	E	F
A						
B						
C	**X**	**X**				
D	**X**	**X**				
E	**X**	**X**				
F			**X**	**X**	**X**	

Minimization of DFA

Step 3: – If there is an unmarked pair (P, Q), such that {δ(P , a), δ (Q , a)} is marked then mark (P,Q).

	A	B	C	D	E	F
A						
B						
C	X	X				
D	X	X				
E	X	X				
F	X	X	X	X	X	

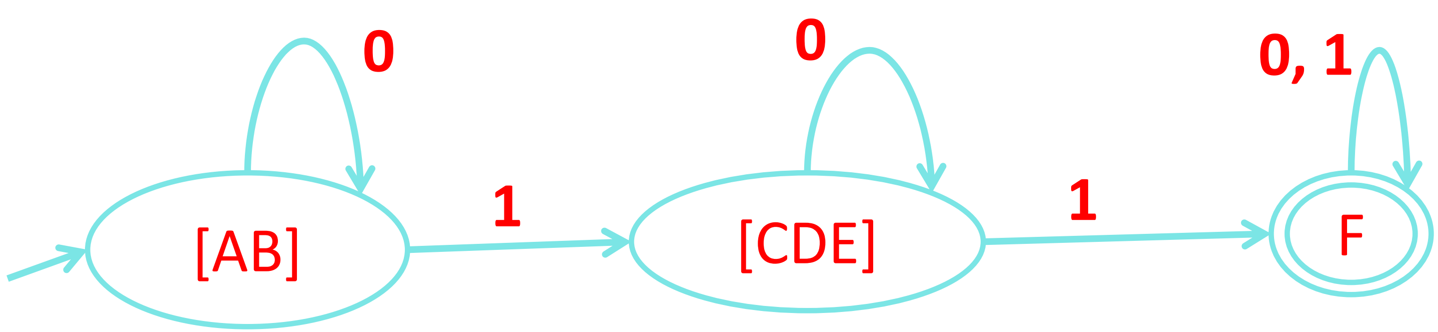
Minimization of DFA

•After step 3, we have got state combinations {a, b} {c, d} {c, e} {d, e} that are unmarked.

•We can recombine {c, d} {c, e} {d, e} into {c, d, e}.

•Hence we got two combined states as − {a, b} and {c, d, e}.

•So the final minimized DFA will contain three states {f}, {a, b} and {c, d, e}.



Finite Automata with Output

- The Finite Automata Discussed so far have limited capability i.e. accepting or rejecting a string.
- Finite Automata with output do not have final state.
- Finite Automata with output are of two types:
    - **Mealy Machine**
    - **Moore Machine**

Finite Automata with Output

- **Mealy Machine**
  - The out put is associated with transition. The output depends on present state and present input.

$$\lambda: Q \times \Sigma \rightarrow \Delta$$

- **Moore Machine**
  - The output is associated with present state. The output depends on present state only.

$$\lambda: Q \rightarrow \Delta$$

Formal Definition of Mealy Machine

- **Mealy machine** is described by 6-tuples - (Q, Σ, Δ, δ, λ, q0)

where

Q = Finite non-empty set of states;

Σ = Set of input alphabets.

Δ = Set of output alphabets.

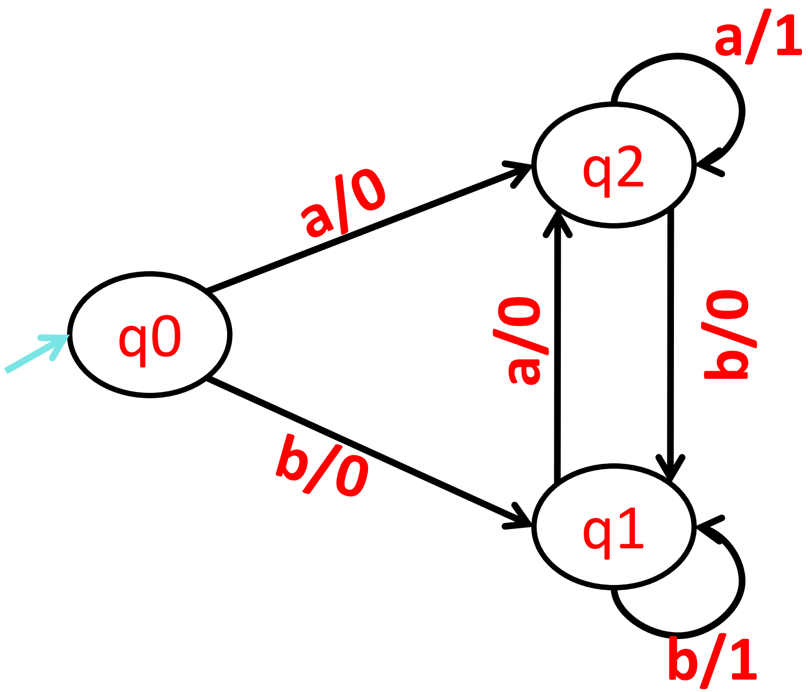
δ = Transitional function mapping Q X Σ → Q

λ = Output function mapping **Q X Σ → Δ**

q0 = Initial state

Example of Mealy Machine

Design a Mealy machine that accepts all strings over ∑={a, b} ending in aa or bb.



Present State	Next State	
	a	b
→q0	q2, 0	q1, 0
q1	q2, 0	q1, 1
q2	q2, 1	q1,0

Formal Definition of Moore Machine

- **Moore machine** is described by 6-tuples - (Q, Σ, Δ, δ, λ, q0)

  where

  Q = Finite non-empty set of states;

  Σ = Set of input alphabets.

  Δ = Set of output alphabets.

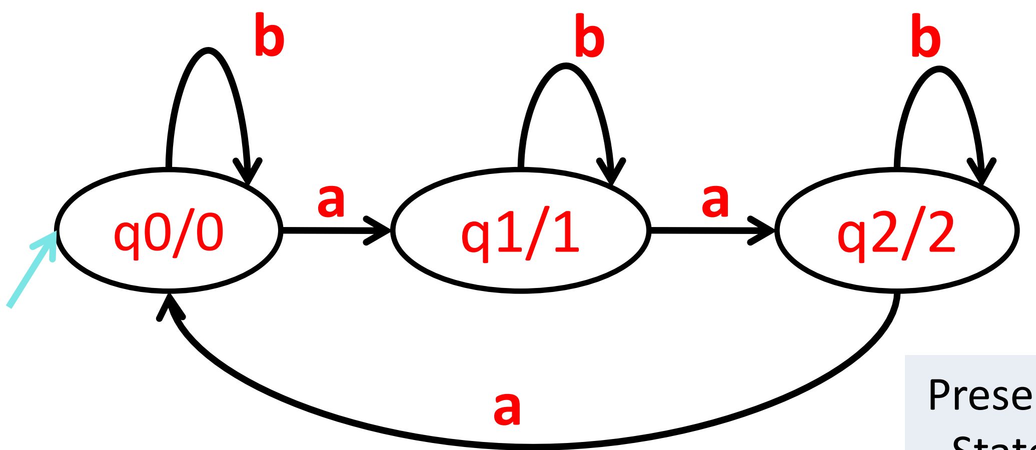
  δ = Transitional function mapping Q X Σ → Q

  λ = Output function mapping **Q → Δ**

  q0 = Initial state

Example of Moore Machine

Create a Moore machine that counts number of **a mod3**. ∑={a, b}.



Present State	Next State		Output
	a	b	
→q0	q1	Q0	0
q1	q2	q1	1
q2	q2	q1	2

Conversion from Mealy machine to Moore machine

Step 1: For each state q determine the number of outputs that are associated with q in Next state column of transition table of the Mealy machine.

Step 2: If the outputs corresponding to state q in the next state columns are same, then retain state q as it is.

Else, break q into different states with the number of new states being equal to the number of different outputs of q.

Step 3: Rearrange the states and outputs in the format of Moore machine.

Conversion from Mealy machine to Moore machine

Step 4: If the output in the constructed state table corresponding to the initial state is 1, then this specifies the acceptance of the null string Ɛ by Mealy machine. Hence, to make both the Mealy and Moore machines equivalent, we either need to ignore the corresponding to null string or we need to insert a new initial state at beginning whose output is 0; the other row elements in this case would remain the same.

Conversion from Mealy machine to Moore machine

Convert given Mealy machine to Moore machine.

Mealy Machine

Present State	Next State	
	a	b
→q0	q3, 0	q1, 1
q1	q0, 1	q3, 0
q2	q2, 1	q2, 0
q3	q1, 0	q0, 1

Moore Machine

Present State	Next State		Output
	a	b	
→q0	q3	q11	1
q10	q0	q3	0
q11	q0	q3	1
q20	q21	q20	0
q21	q21	q20	1
q3	q10	q0	0

Split **q1** into **q10** and **q11**

and

Split **q2** into **q20** and **q21**

Conversion from Moore machine to Mealy machine

For understanding the conversion of Moore machine to Mealy machine, let us take an example:

Suppose the Moore machine transition table is:

Present State	Next State		Output
	a	b	
→p	s	q	0
q	q	r	1
r	r	s	0
s	s	p	0

Conversion from Moore machine to Mealy machine

First of all take the Mealy machine transition table format, and copy all the Moore machine transition table states into Mealy machine transition table.

Present State	Next State	
	a	b
→p	s	q
q	q	r
r	r	s
s	s	p

Conversion from Moore m/c to Mealy m/c

Now in the Moore machine, the output of state p is 0. So make the output of p in the Mealy machine next state column of the above table is 0. Same process is repeated for q, r and s.

Moore Machine

Present State	Next State		Output
	a	b	
→p	s	q	0
q	q	r	1
r	r	s	0
s	s	p	0

Mealy Machine

Present State	Next State	
	a	b
→p	s, 0	q, 1
q	q, 1	r, 0
r	r, 0	s, 0
s	s, 0	p, 0

NIET Greater Noida
GET FUTURE READY
Estd 2001
AN AUTONOMOUS INSTITUTE

Difference Between Mealy And Moore Machine

BASIS OF COMPARISON	MEALY MACHINE	MOORE MACHINE
Description	Mealy machine changes its output based on its current input and present state.	Output of Moore machine only depends on its current state and not on the current input.
Output	Output is placed on transition.	Output is placed on transition.
Output Function	$\lambda: Q \times \Sigma \rightarrow \Delta$	$\lambda: Q \rightarrow \Delta$
Counter	A counter is not a Mealy machine.	A counter is a Moore machine.
Design	Not necessarily easy to design.	Easy to design.
Length of Out put String	It produce n length output string corresponding to n length input String	It produce n+1 length output string corresponding to n length input String

Video Links

- NPTEL Video Links

  https://youtu.be/al4AK6ruRek

  https://youtu.be/539Bk9fFOyo

  https://youtu.be/r20I_inUNv8

Daily Quiz

1. Given: ∑= {a, b}L= {x∈∑* | x is a string combination}∑4 represents which among the following? *

A. {aa, ab, ba, bb}
B. {aaaa, abab, ε, abaa, aabb}
C. {aaa, aab, aba, bbb}
D. All of the mentioned

2. Converting each of the final states of F to non-final states and old non-final states of F to final states, FA thus obtained will reject every string belonging to L and will accept every string, defined over Σ, not belonging to L. is called

A. Transition Graph of L
B. Regular expression of L
C. Complement of L
D. Finite Automata of L

Daily Quiz

3. Myhill Nerode theorem is consisting of the followings,

A. L partitions Σ into distinct classes.

B. If L is regular then, L generates finite number of classes.

C. If L generates finite number of classes, then L is regular.

D. All of above

4. The part of an FA, where the input string is placed before it is run, is called ______

A. State

B. Transition

C. Input Tape

D. Output Tape

Daily Quiz

5. Which of the following is an application of Finite Automaton?

A. Compiler Design
B. Grammar Parsers
C. Text Search
D. All of the mentioned

6. Which of the following is a not a part of 5-tuple finite automata?

A. Input alphabet
B. Transition function
C. Initial State
D. Output Alphabet

Daily Quiz

7. John is asked to make an automaton which accepts a given string for all the occurrence of '1001' in it. How many number of transitions would John use such that, the string processing application works?

A. 9
B. 11
C. 12
D. 15

8. The total number of states to build the given language using DFA:
L= {w | w has exactly 2 a's and at least 2 b's}

A. 10
B. 11
C. 12
D. 13

Daily Quiz

9. A binary string is divisible by 4 if and only if it ends with:

a) 100

b) 1000

c) 1100

d) 0011

10. Let N (Q, ∑, δ, q0, A) be the NFA recognizing a language L. Then for a DFA (Q', ∑, δ', q0', A'), which among the following is true?

a) Q' = P(Q)

b) Δ' = δ' (R, a) = {q ϵ Q | q ϵ δ (r, a), for some r ϵ R}

c) Q'={q0}

d) All of the mentioned

Daily Quiz

1. One language can be expressed by more than one FA". This statement is ______

a) True
b) False
c) Some times true & sometimes false
d) None of these

2. Can a DFA simulate NFA?

a) NO
b) YES
c) SOMETIMES
d) Depends on NFA

Daily Quiz

3. Which of the following statements is wrong ?

A. The language accepted by finite automata are the languages denoted by regular expressions
B. For every DFA there is a regular expression denoting its language
C. For a regular expression r, there does not exist NFA with L(r) any transit that accept
D. None of these

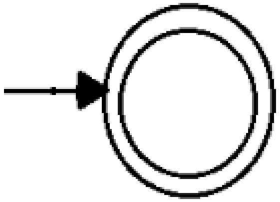
4. An automation is a __________ device and a grammar is a __________ device.
A. generative, cognitive
B. generative, acceptor
C. acceptor, cognitive
D. cognitive, generative

Daily Quiz

5. Finite state machines __________ recognize palindromes

A. can
B. can't
C. may
D. may not

6. FSM shown in the figure



A. all strings
B. no string
C. ε- alone
D. none of these

Daily Quiz

7. A FSM can be used to add how many given integers?

a) 1
b) 2
c) 3
d) Any number of integers

8. The basic limitation of a FSM is that

a) It cannot remember arbitrary large amount of information
b) It sometimes recognizes grammar that are not regular
c) It sometimes fails to recognize grammars that are regular
d) All of the above

Daily Quiz

9. Finite automata are used for pattern matching in text editors for

a) Compiler lexical analysis
b) Programming in localized application
c) Both A and B
d) None of the above

10. The language accepted by finite automata is

a) Context free
b) Regular
c) Non regular
d) None of these

MCQs

1. A binary string is divisible by 4 if and only if it ends with:

A. 100

B. 1000

C. 1100

D. 0011

2. Recognizing capabilities of NFSM and DFSM

A. May be different

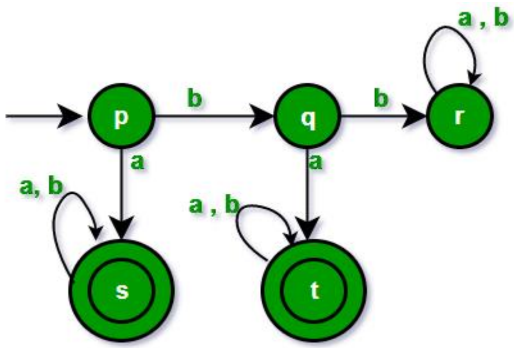
B. May be same

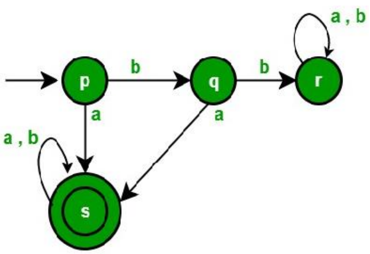
C. Must be different

D. None of the above

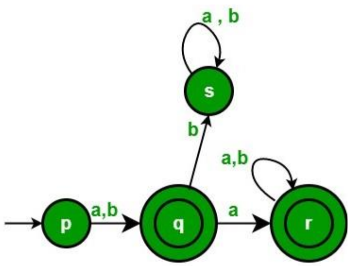
MCQs

3. Which minimum state FA is equivalent to following FA

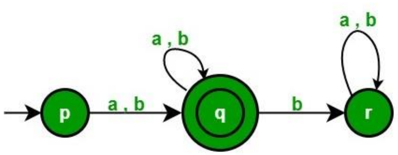




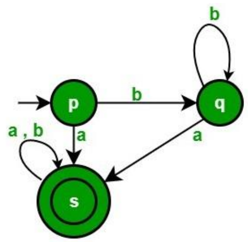
(A)



(B)



(C)



(D)

ANS : (A)

MCQs

4. The minimum number of states required to recognize an octal number divisible by 3 are/is

A. 1

B. 3

C. 5

D. 7

5. Which of the following is/ are regular

A. a string of a' s in perfect square

B. a string of palindrom over {a,b}

C. a string of odd no of a's over {a,b}

D. a string of equal no of a's and b's over {a,b}

MCQs

6. If two finite state machines are equivalent, they should have the same number of

A. states

B. edges

C. states and edges

D**. none of these**

7. The word 'formal' in formal languages means

A. the symbols used have well-defined meaning

B. they are unnecessary, in reality

C. only form of the string of symbols is significant

D. Both (a) and (b)

MCQs

8. The main difference between a DFSA and an NDFSA is

A. in DFSA, ε transition may be present

B. in NDFSA, ε transitions may be present

C. in DFSA, from any given state, there can't be any alphabet leading to two diferent states

D. in NDFSA, from any given state, there can't be any alphabet leading to two different states

9. Palindromes can't be recognized by any FSM because

A. FSM can't remember arbitrarily large of information

B. FSM can't deterministically fix the mid-point

C. even if mid-point is known, FSM be can't be found whether, second half of the string matches the first half

D. all of these

Glossary

1) Construct the DFA for the set of all string over {a,b} contain exactly 3a's .
2) Construct the DFA that accepts all string over{a,b} contains at most 3 a's
3) Construct the DFA for the even number of a's over {a,b}.
4) Construct the DFA for the set of all string contain three consecutive a's over {a,b}.
5) Construct the DFA that accepts all string over{a,b} containing aba as substring.

Weekly Assignment

1. Design a deterministic finite automaton(DFA) for the following language over the set of input alphabet {0,1}

[CO1]

- All strings of 0’s and 1’s such that no of 0’s are even and 1’s are odd.
- All strings of 0’s and 1’s with at least two consecutive 0’s.
- All strings of 0’s and 1’s beginning with 1 and not having two consecutive zeroes.
- All strings of 0’s and 1’ s not containing 101 as substring.
- All strings of 0’s and 1’s whose last two symbols are same.

Weekly Assignment

2. Design a Non-deterministic finite automaton(NFA) for the following language over the set of input alphabet {0,1}

[CO1]

- All strings of 0's and 1's such that 3rd symbol from right end is 1.
- All strings of 0's and 1's such that either the 2nd or 3rd position from the right end has a 1.
- All strings of 0's and 1's satisfying 1^m01^n : m, n>=1.
- All strings of 0's, 1's and 2's with any no of 0's followed by any no of 1's and any no of 1's followed by any no of 2's.
- All strings of 0's and 1's ending in1 and not containing substring 00.

Weekly Assignment

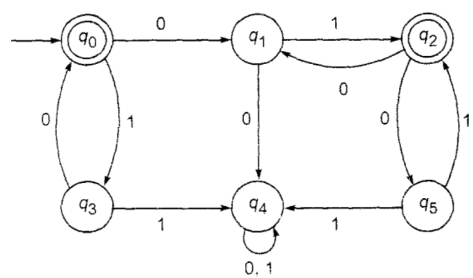
3. Prove that NFA is equivalent to DFA. **[CO1]**
4. Construct NFA accepting the set of all strings over {a, b} ending in aba. Use it to construct a DFA accepting the same set of strings. **[CO1]**
5. Design a NFA with epsilonthat accepts {a, b}*baaa.**[CO1]**
6. Design a NFA that accepts (a+b)*(ab+bba) (a+b)* i.e. strings containing either ab or bba as substring . convert it into DFA. **[CO1]**
7. Differentiate between DFA and NDFA with suitable example? **[CO1]**
8. Describe various Application and Limitations of Finite Automata. **[CO1]**

Old Question Papers

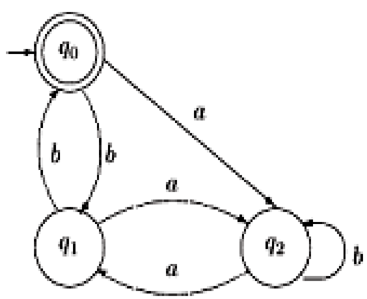
https://drive.google.com/drive/folders/19Eia3VHCl3627foiH6V_j-p4X9ZkyyC7?usp=sharing

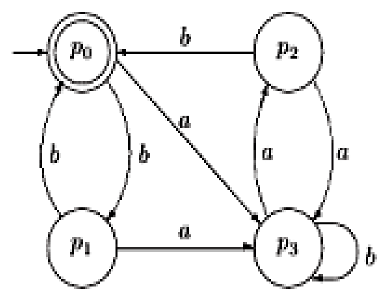
Expected Questions for University Exam

- Design a NFA that accepts all the strings for input alphabet {a,b} containing the substring abba.
- Convert NFA into equivalent DFA by taking any suitable example.
- Design the DFA that accepts an even number of a's and even number of b's.
- Construct the minimum state automata equivalent to DFA described below:



- Check with the comparison method for testing equivalence of two FA given below:





Recap

- Finite automata is a machine that acccepts regular languages.
- FA has its application in many fields like compiler design, digital circuits, etc.
- NFA and DFA has same expressive power.
- NFA is easy to construct than DFA.
- Every NFA is equivalent to DFA.
- Myhill-Nerode theorem is used to optimize the FA.

References

- Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2001). Introduction to automata theory, languages, and computation. *Acm Sigact News*, *32*(1), 60-65.
- Linz, P. (2006). *An introduction to formal languages and automata*. Jones & Bartlett Learning.
- Mishra, K. L. P., & Chandrasekaran, N. (2006). *Theory of Computer Science: Automata, Languages and Computation*. PHI Learning Pvt. Ltd..

Thank You