Noida Institute of Engineering and Technology, Greater Noida

Theory of Automata and Formal Languages (KCS402)

Unit: I

Finite Automata

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

Twinkle Tyagi
Department of Computer Science & Engineering

Index

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.

Program Specific Outcomes

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

PSO1:	Design and develop the Hardware and Software systems.
PSO2:	Understand the interdisciplinary computing techniques and an ability to apply them in the design of advanced computing.
PSO3:	Understand the programming methodology, software development paradigms, design and analysis of Algorithms, Operating Systems, Digital Logic Design, Theory of Computation, Discrete Mathematics, Compiler Design, etc.
PSO4:	To integrate & manage the various phases/components of software development projects of society.

Course Outcomes

CO	At the end of course , the student will be able to	Bloom's (KL)
CO1	Analyze, design finite automata, pushdown automata, Turing machines, formal languages, and grammars	K_4, K_5
CO2	Analyze and design, Turing machines, formal languages, and grammars	K_4, K_5
CO3	Demonstrate the understanding of key notions, such as algorithm, computability, decidability, and complexity through problem solving	K_1, K_5
CO4	Prove the basic results of the Theory of Computation	K_2, K_3
CO5	State and explain the relevance of the Church-Turing thesis	K_1, K_5

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-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

Syllabus

UNIT-1

Basic Concepts and Automata Theory: Introduction to Theory of Computation- Automata, Computability and Complexity, Alphabet, Symbol, String, Formal Languages,

Deterministic Finite Automaton (DFA)- Definition, Representation, Acceptability of a String and Language,

Non Deterministic Finite Automaton (NFA), Equivalence of DFA and NFA, NFA with ε-Transition, Equivalence of NFA's with and without ε-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

Syllabus

UNIT-2

Regular Expressions and Languages: Regular Expressions, Transition Graph, Kleen’s Theorem, Finite Automata and Regular Expression- Arden’s theorem, Algebraic Method Using Arden’s Theorem,

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, Regular Languages and Computers, Simulation of Transition Graph and Regular language.

Syllabus

UNIT-3

Regular and Non-Regular Grammars: Context Free Grammar(CFG)-Definition, Derivations, Languages, Derivation Trees and Ambiguity,

Regular Grammars-Right Linear and Left Linear grammars,

Conversion of FA into CFG and Regular grammar into FA,

Simplification of CFG, Normal Forms- Chomsky Normal Form(CNF), Greibach Normal Form (GNF),

Chomsky Hierarchy, Programming problems based on the properties of CFGs.

Syllabus

UNIT-4

Push Down Automata and Properties of Context Free Languages: Nondeterministic Pushdown Automata (NPDA)- Definition, Moves, A Language Accepted by NPDA, Deterministic Pushdown Automata(DPDA) and Deterministic Context free Languages(DCFL), Pushdown Automata for Context Free Languages,

Context Free grammars for Pushdown Automata,

Two stack Pushdown Automata,

Pumping Lemma for CFL,

Closure properties of CFL,

Decision Problems of CFL, Programming problems based on the properties of CFLs.

Syllabus

UNIT-5

Turing Machines and Recursive Function Theory : Basic Turing Machine Model, Representation of Turing Machines, Language Acceptability of Turing Machines, Techniques for Turing Machine Construction, Modifications of Turing Machine, Turing Machine as Computer of Integer Functions,

Universal Turing machine, Linear Bounded Automata,

Church’s Thesis,

Recursive and Recursively Enumerable language,

Halting Problem,

Post’s Correspondance Problem,

Introduction to Recursive Function Theory.

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

UNIT-1

Contents

Topics	Duration (in Hours)
Introduction to theory of computation	1
Finite Automata	6
Minimization of DFA	2
Finite Automata with Output	2

Objectives of the Unit

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.

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

CO-PSO correlation matrix

CO	PSO			
	PSO1	PSO2	PSO3	PSO4
CO1	2	2	2	2

Prerequisite for the Course

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

Finite Automata

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

Finite Automata

Topic mapping with Course Outcome

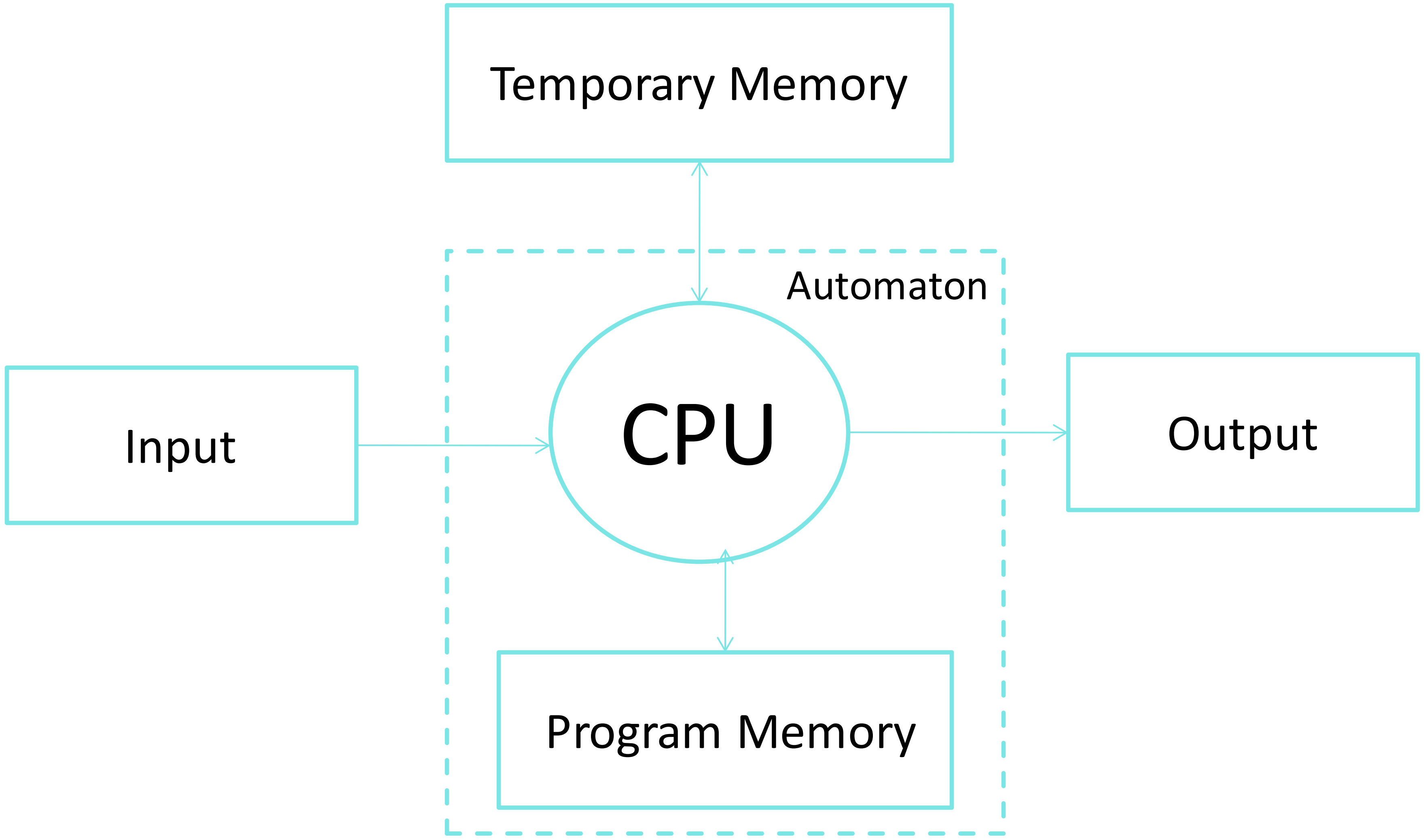
Topic	CO1	CO2	CO3	CO4	CO5
Finite Automata	3	-	-	-	-

Introduction to Automata

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

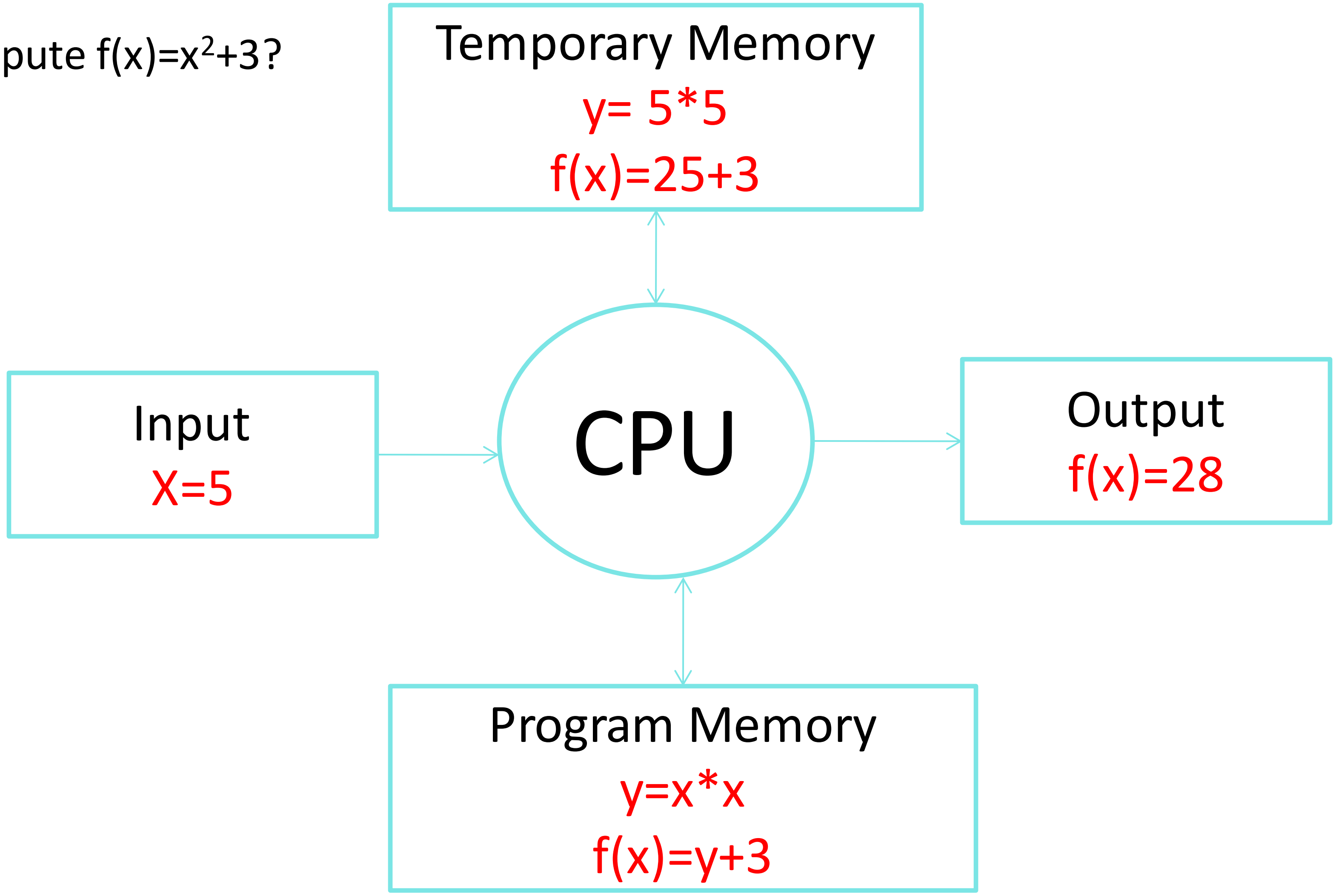
- Automata theory is the study of abstract computational machines and the computational problems that can be solved using these machines
- Abstract machine are (simplified) mathematical models of real computations

Introduction to Automata



Introduction to Automata

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



Introduction to Automata

Different kinds of Automata

On the basis of temporary memory

• **Finite Automata**:	no temporary memory
• **Pushdown Automata**:	stack
• **Turing Machines**:	random access memory

Common terms used in Automata

Alphabet

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

It is represented by ∑.

Ex:

∑={0,1} is binary alphabet.

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

∑={a,b,c,.....,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}.

Common terms used in Automata

Length of a string

or

Power of ∑

Σ^k is the set of strings of length k (|w|=k).

Ex:

If ∑={a,b} then

Σ^0= {Ɛ} , called Epsilon.

Σ^1 = {a,b}

Σ^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

NIET Greater Noida
GET FUTURE READY

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

Common terms used in Automata

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={a,b}* | |w|$\leq$3]

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

Common terms used in Automata

Kleene closure

Kleene closure represented as ∑*, is set of strings of all possible length.

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

Ex:

If ∑ ={a,b} then

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

Common terms used in Automata

Positive closure

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

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

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

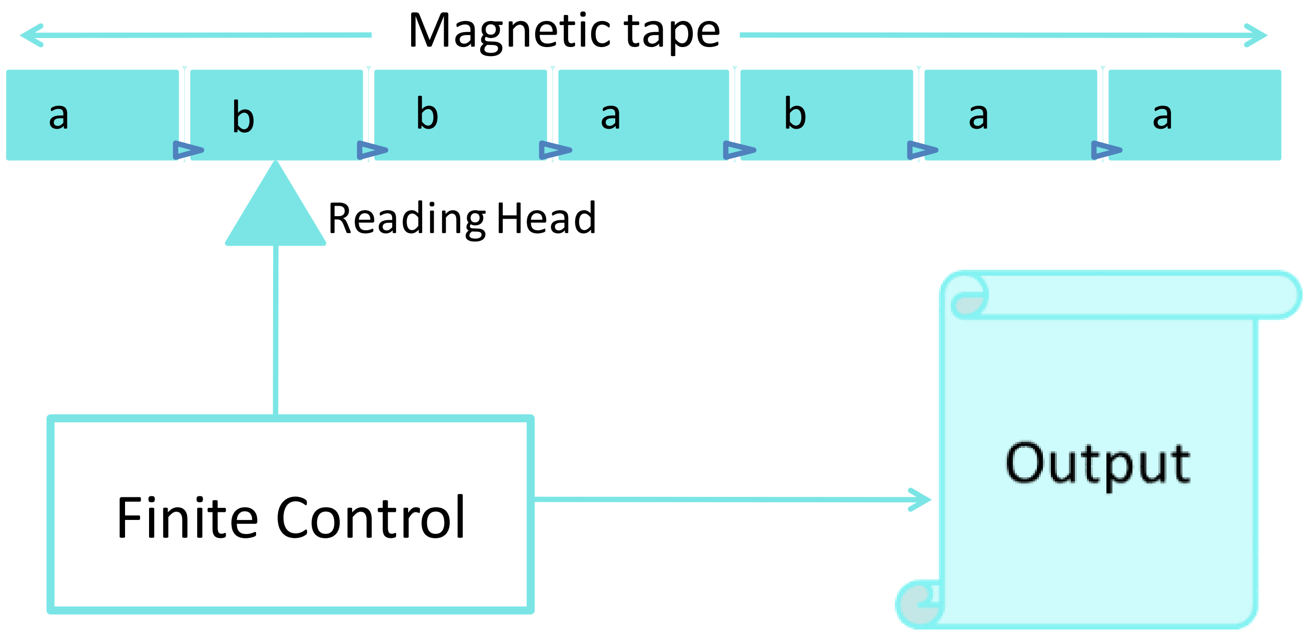
Ex:

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

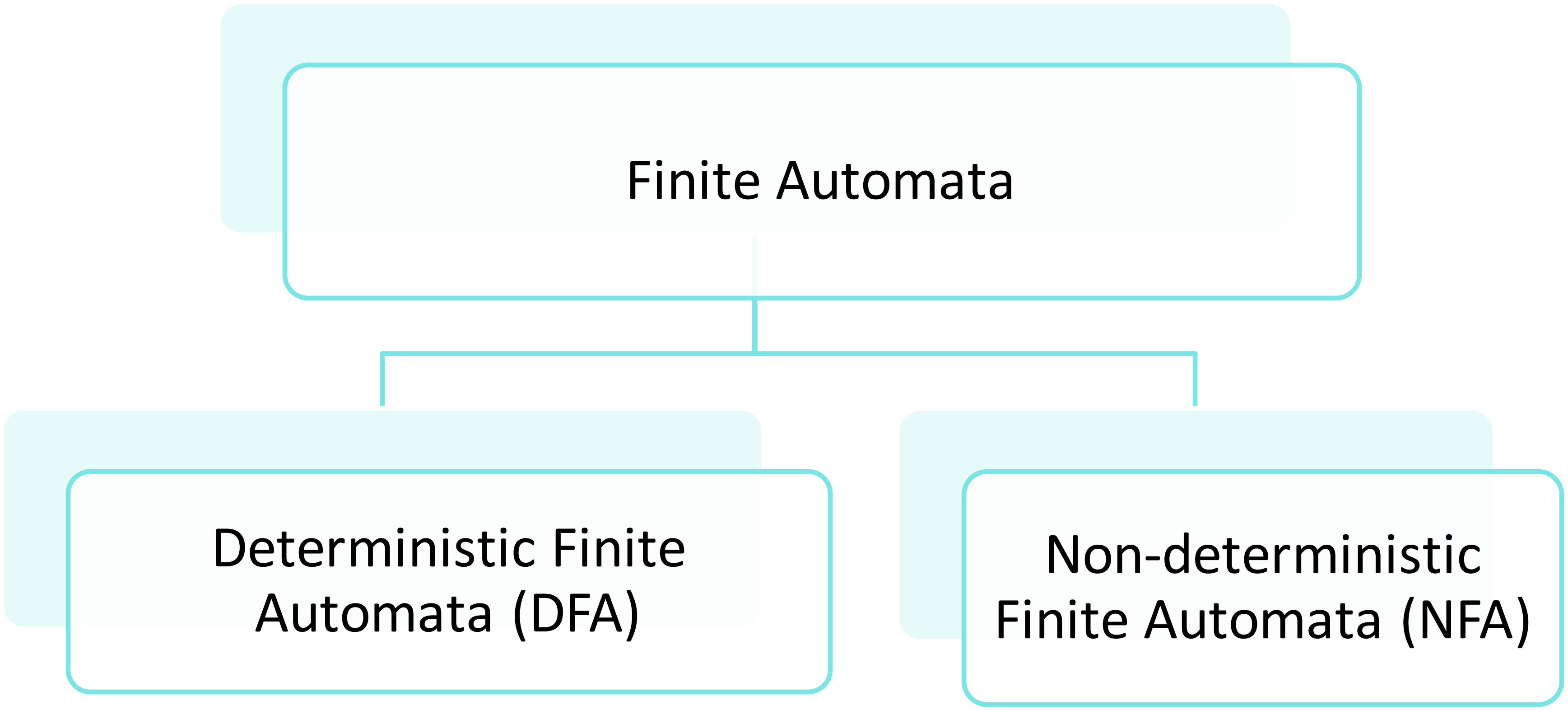
$\sum^+ = \{a,b,aa,ab,ba,bb,aaa,aab,\ldots\ldots\ldots\}$

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.



Finite Automata


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

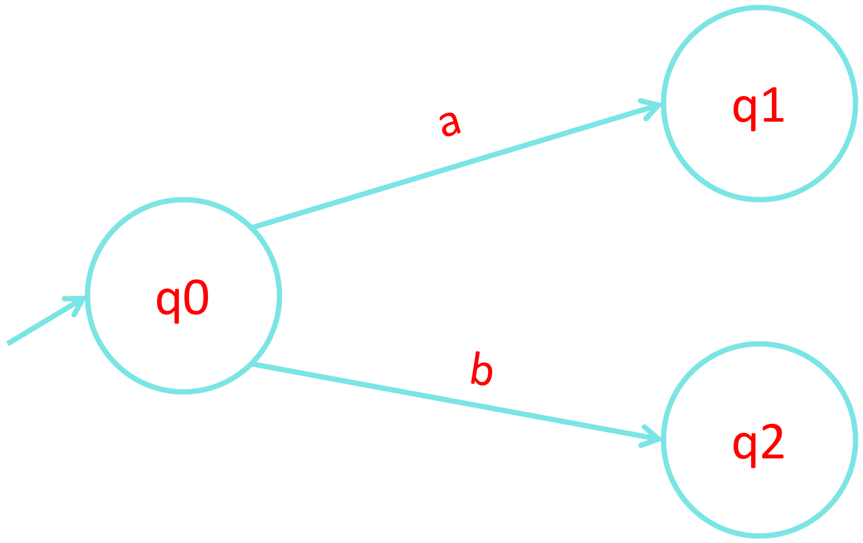
Applications of Automata

Applications of Finite Automata

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

DFA

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.



Formal definition of a DFA

- A DFA 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, $\delta: Q \times \Sigma \rightarrow Q$

  q_0= Initial State

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

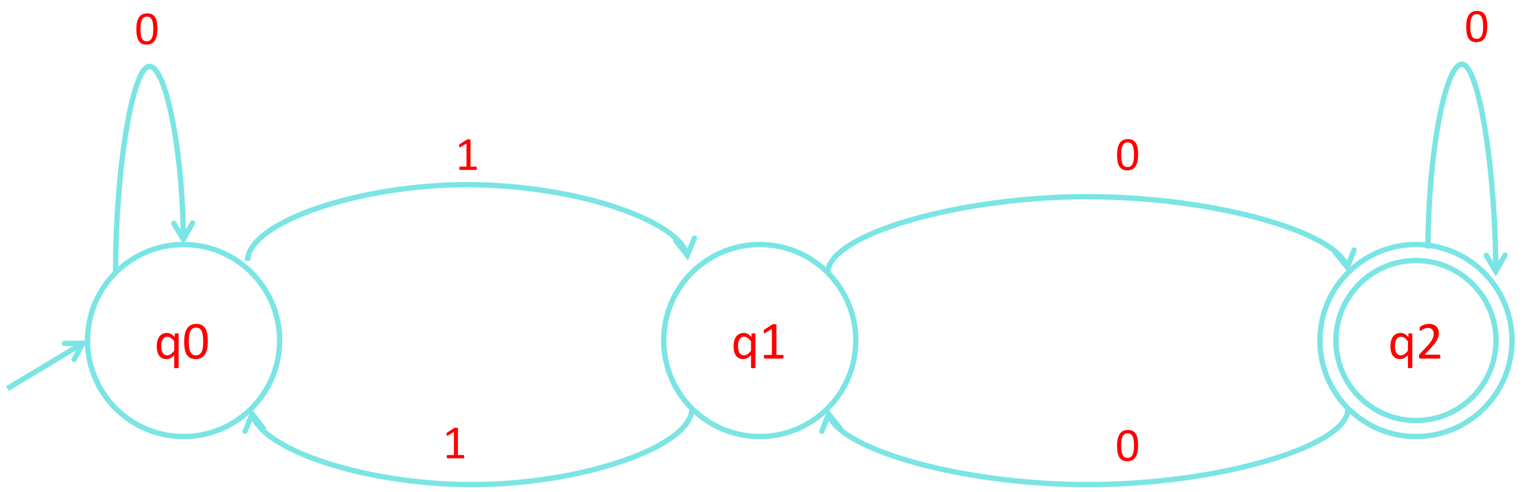
DFA

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	q0
q1	q0	q0
*q2	q0	q0

DFA

Transition Function

$\delta(q_0, 0) = q_0$

$\delta(q_0, 1) = q_1$

$\delta(q_1, 0) = q_2$

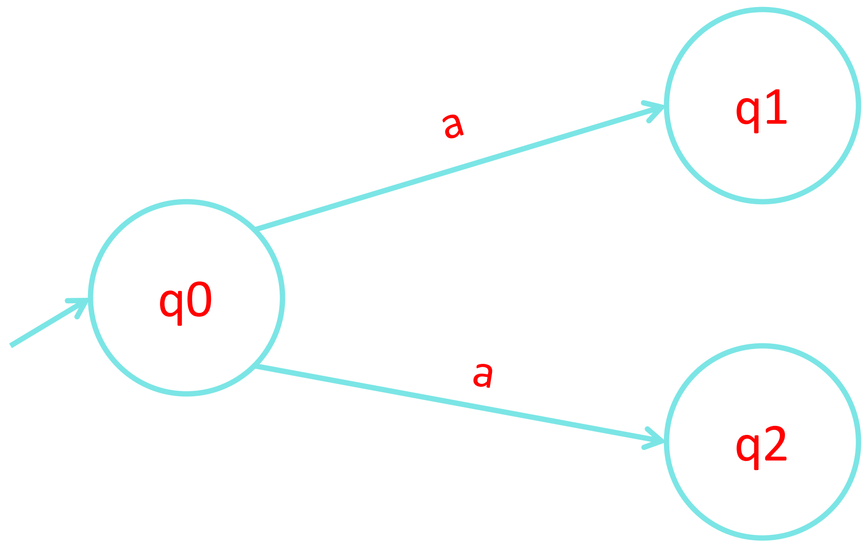
$\delta(q_1, 1) = q_0$

$\delta(q_2, 0) = q_1$

$\delta(q_2, 1) = q_2$

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

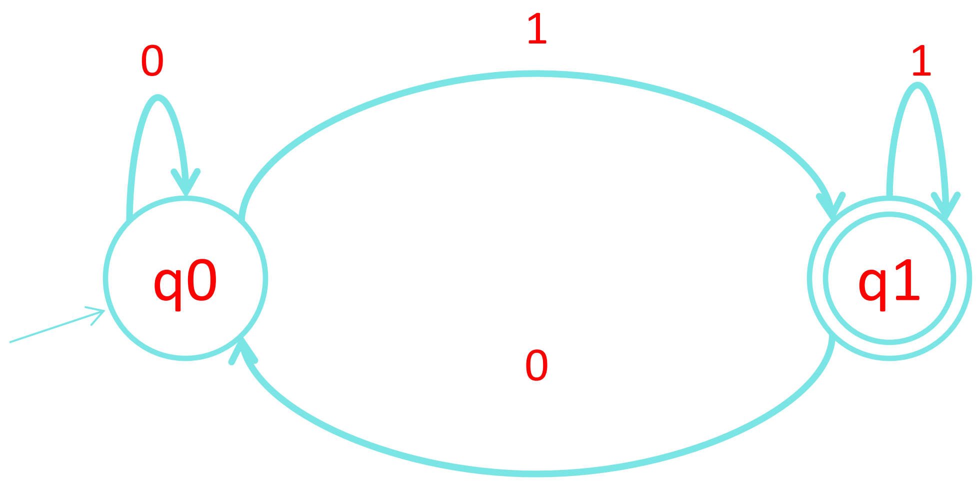
  F= Set of final states ($F \subseteq Q$)

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.
Backtracking is allowed in DFA	In NDFA, backtracking is not always possible.
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.

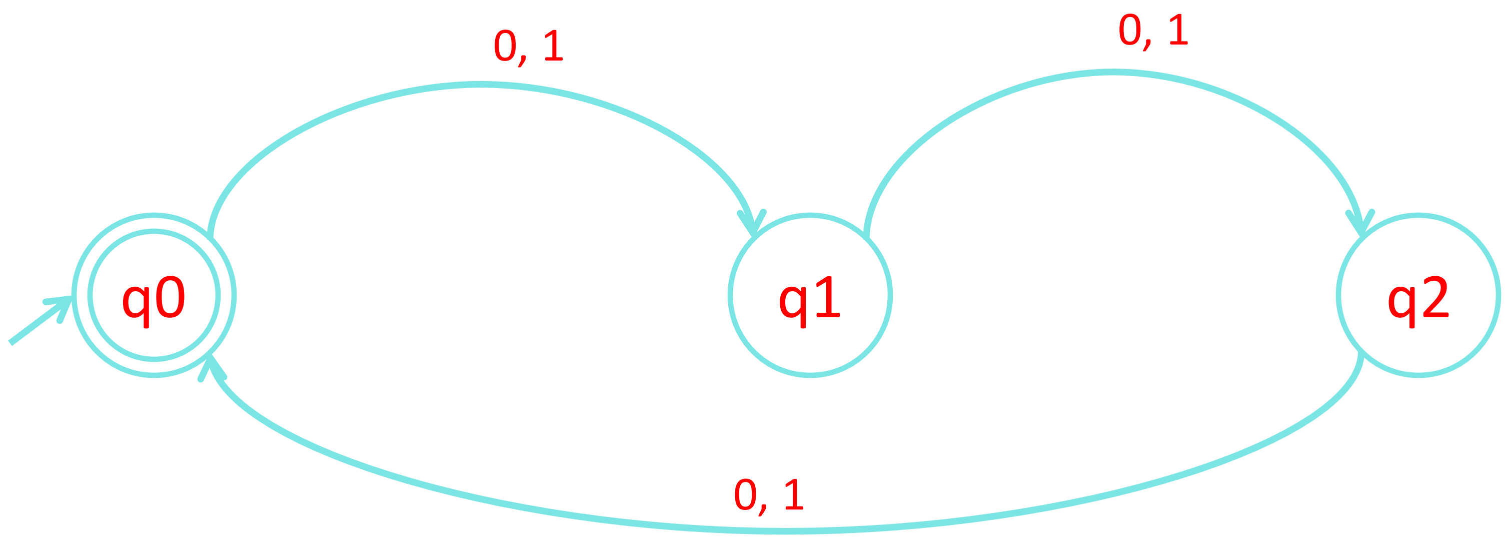
Examples of DFA

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



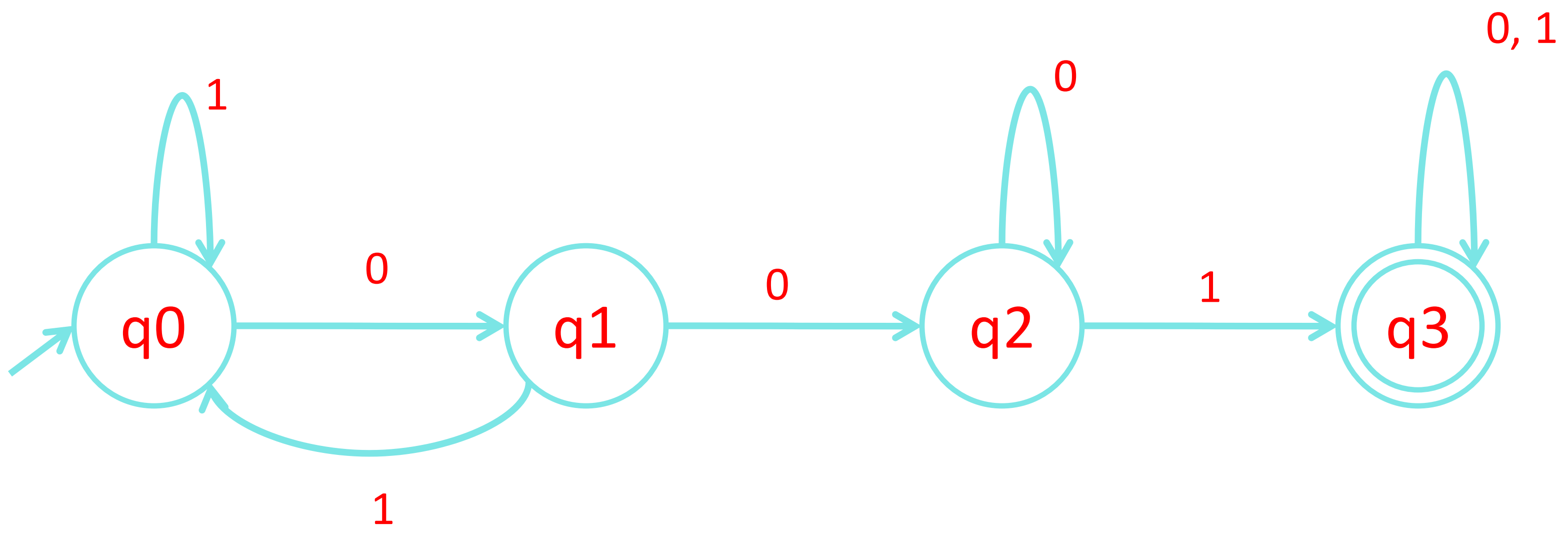
Examples of DFA

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


$$\begin{tikzcd}
\arrow[r] & q0 \arrow[rr, bend left, "{0, 1}"] & & q1 \arrow[rr, bend left, "{0, 1}"] & & q2 \arrow[llll, bend left, "{0, 1}"]
\end{tikzcd}$$

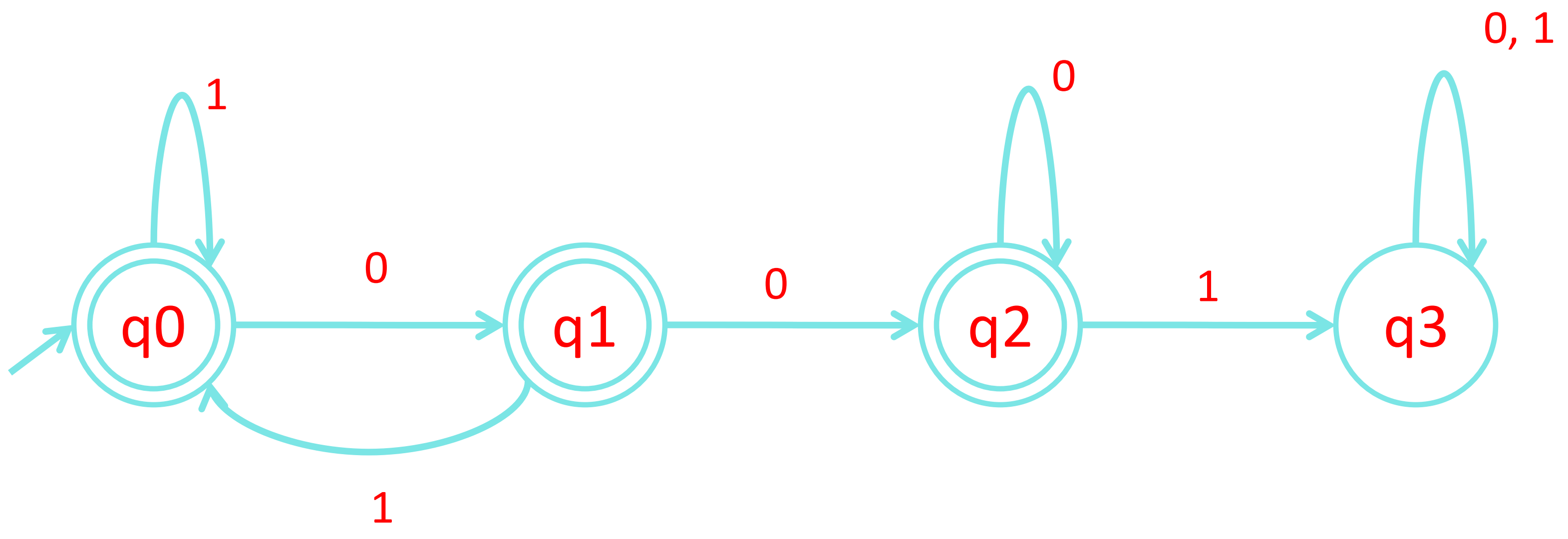
Examples of DFA

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



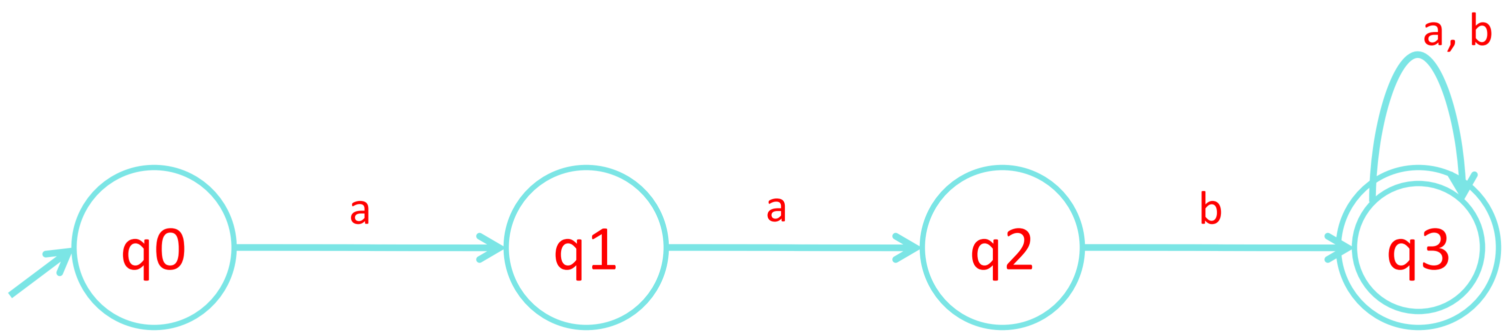
Examples of DFA

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



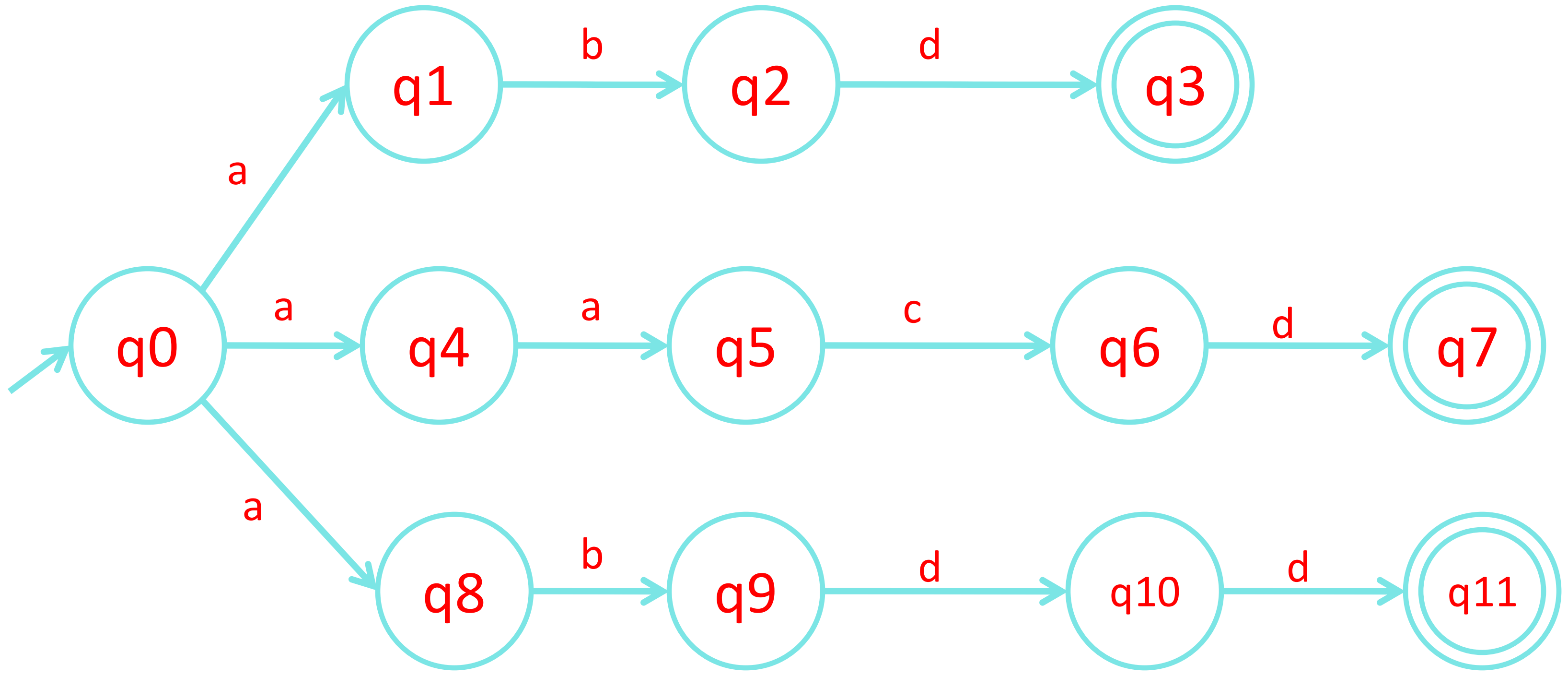
Examples of NFA

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


$$\begin{tikzcd}
\arrow[r] & q0 \arrow[r, "a"] & q1 \arrow[r, "a"] & q2 \arrow[r, "b"] & q3 \arrow[loop above, "{a, b}"]
\end{tikzcd}$$

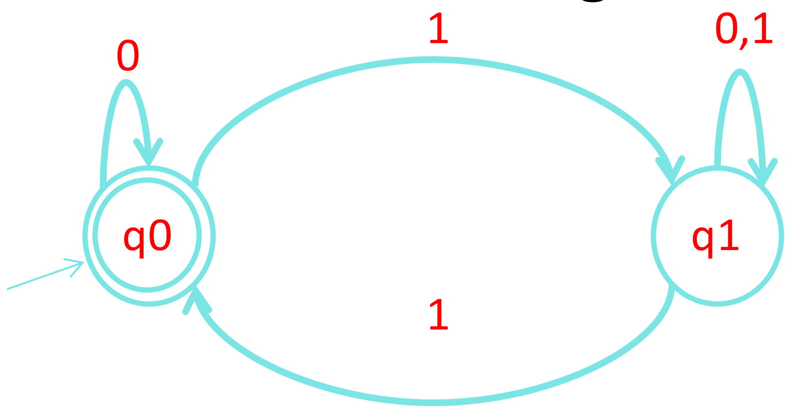
Examples of NFA

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



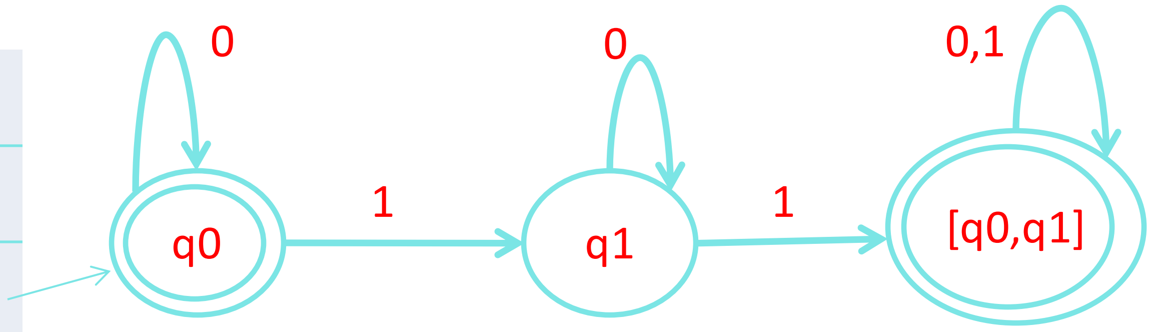
Conversion of NFA to DFA

- 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]



Equivalence of NFA and DFA

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)

Equivalence of NFA and DFA (Proof)

Basis Step

Let *x* be the empty string ε.

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

Equivalence of NFA and DFA (Proof)

Inductive Step

Assume that for any *y* with |*y*| ≥ 0, $\delta_D(q_{0D,}\ y) = \delta_N(q_{0N,}\ y)$.

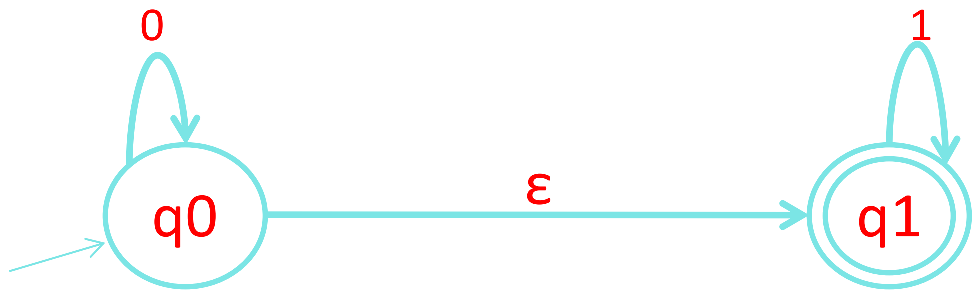
If we let *n* = |*y*|, then we need to prove that for a string *z* with |*z*| = *n* + 1, $\delta_D(q_{0D,}\ z) = \delta_N(q_{0N,}\ z)$. We can represent the string *z* as a concatenation of string *y* (|y| = *n*) and symbol *a* from the alphabet Σ (*a* ∈ Σ). So, *z* = y*a*.

$$\begin{aligned}\delta_D(q_{0D,}\ z) &= \delta_D(q_{0D,}\ ya)\\ &= \delta_D(\delta_N(q_{0D,}\ y)\ ,a)\\ &= \delta_D(\delta_N(q_{0N,}\ y)\ ,a) && \text{By assumption}\\ &= \cup_{p\in\ \delta N(q0N,\ y)}\ \delta_N(q_{0N,}\ y) && \text{By definition of } \delta_D\\ &= \delta_N(q_{0N,}\ ya)\\ &= \delta_N(q_{0N,}\ z)\end{aligned}$$

DFA D accepts a string *x* iff $\delta_D(q_{0D,}\ x) \in F_D$. From the above it follows that D accepts *x* iff $\delta_N(q_{0N,}\ x) \cap F_N = \phi$.

So a string is accepted by DFA D if, and only if, it is accepted by NFA N.

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, ∑, δ, q_0, F}

  Q= Non empty finite set of states

  ∑= Input alphabet

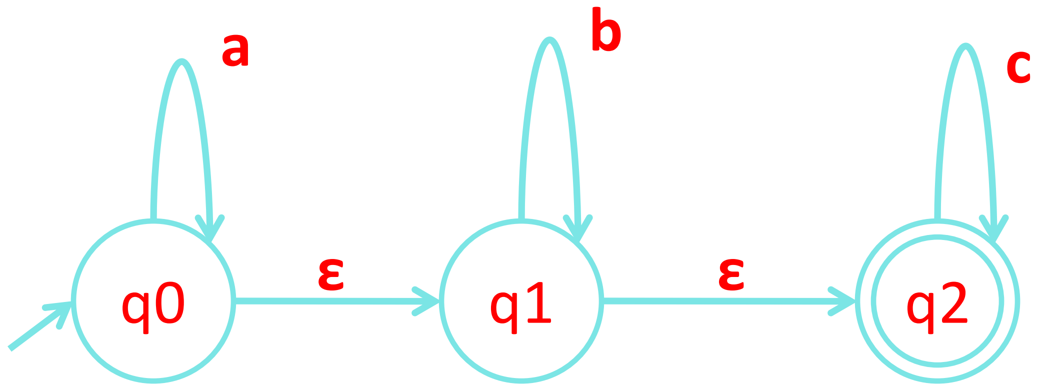
  δ= Transition Function, $\delta: Q \times (\Sigma \cup \varepsilon) \rightarrow 2^Q$

  q_0= Initial State

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

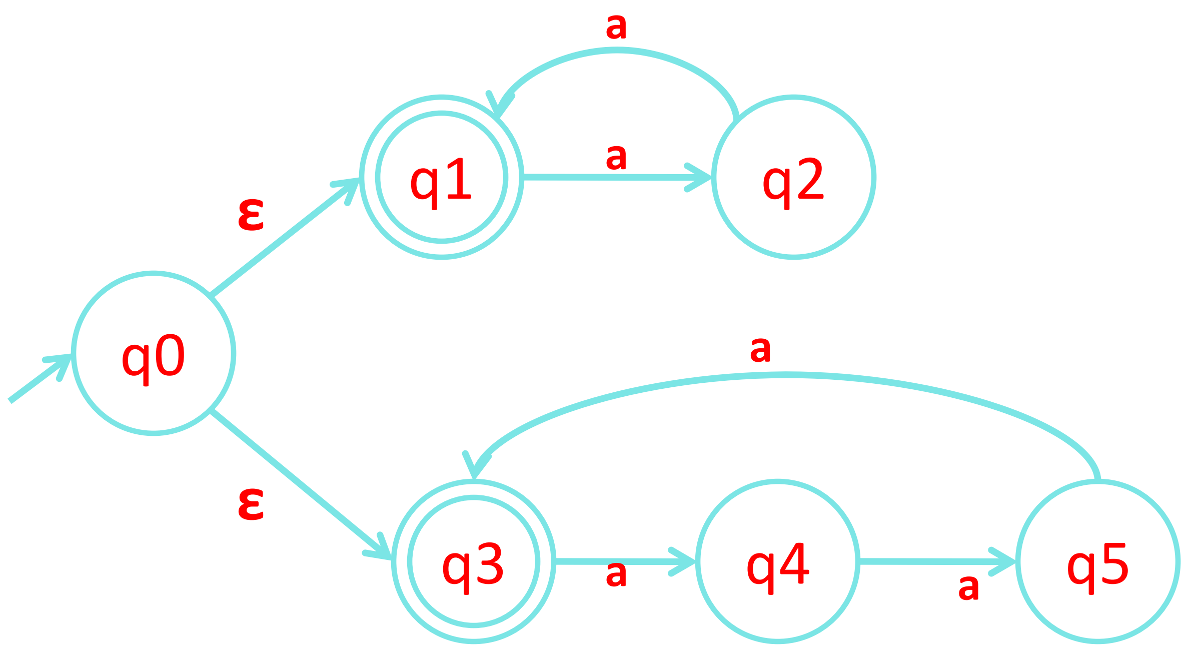
Example of ε -NFA

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


$$\begin{tikzcd} \arrow[r] & q0 \arrow[loop above, "a"] \arrow[r, "\varepsilon"] & q1 \arrow[loop above, "b"] \arrow[r, "\varepsilon"] & q2 \arrow[loop above, "c"] \end{tikzcd}$$

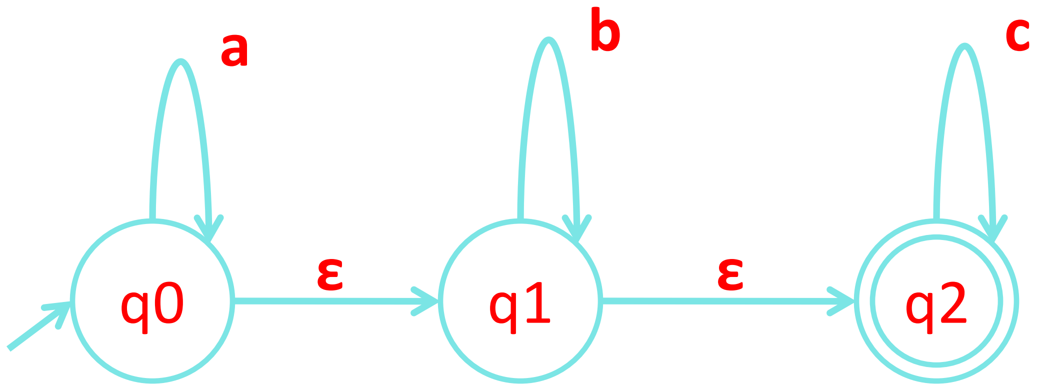
Example of ε -NFA

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



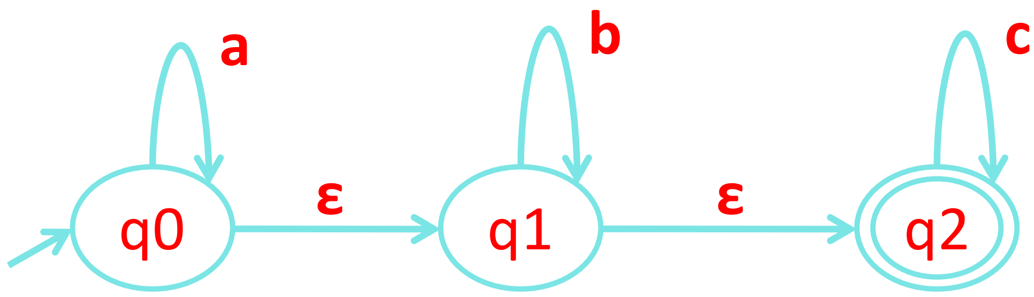
ε –Closure of a state

- 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}

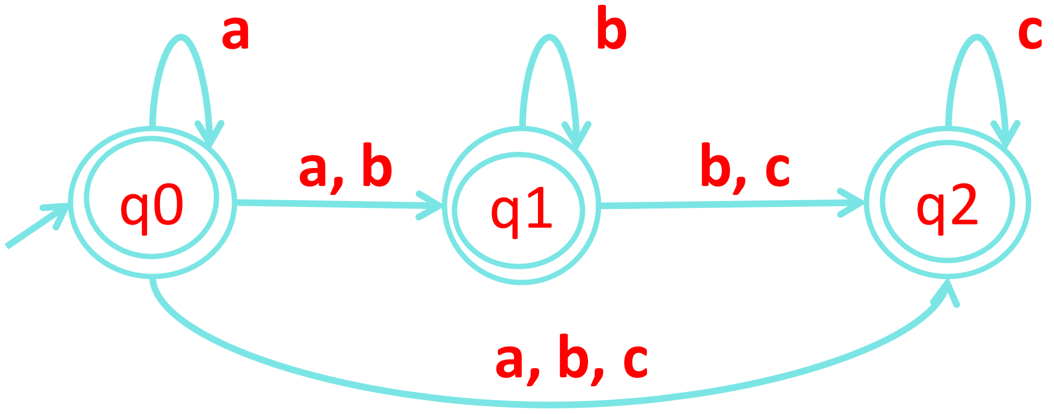
Conversion of ε –NFA to NFA



$\delta(q0, a) = \varepsilon\text{-Closure}\,\delta(\varepsilon\text{-Closure}(q0), a)$
$= \varepsilon\text{-Closure}\,\delta(\{q0, q1, q2\}, a)$
$= \varepsilon\text{-Closure}(q0)$
$= \{q0, q1, q2\}$

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}



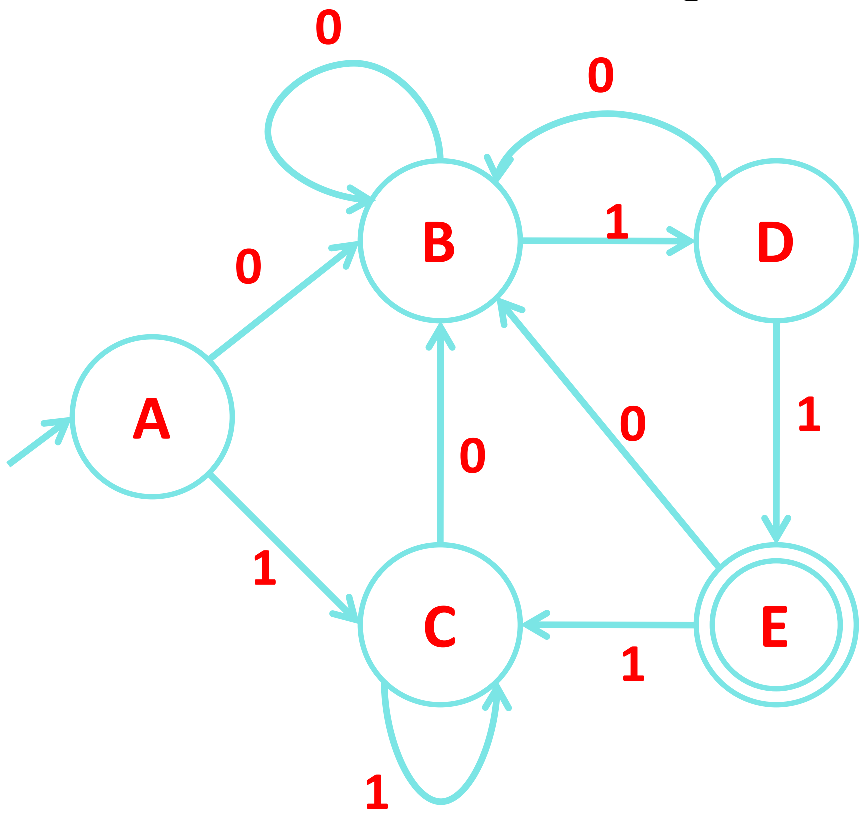
Minimization of DFA

- 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

Transition Table

	0	1
$\rightarrow$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}

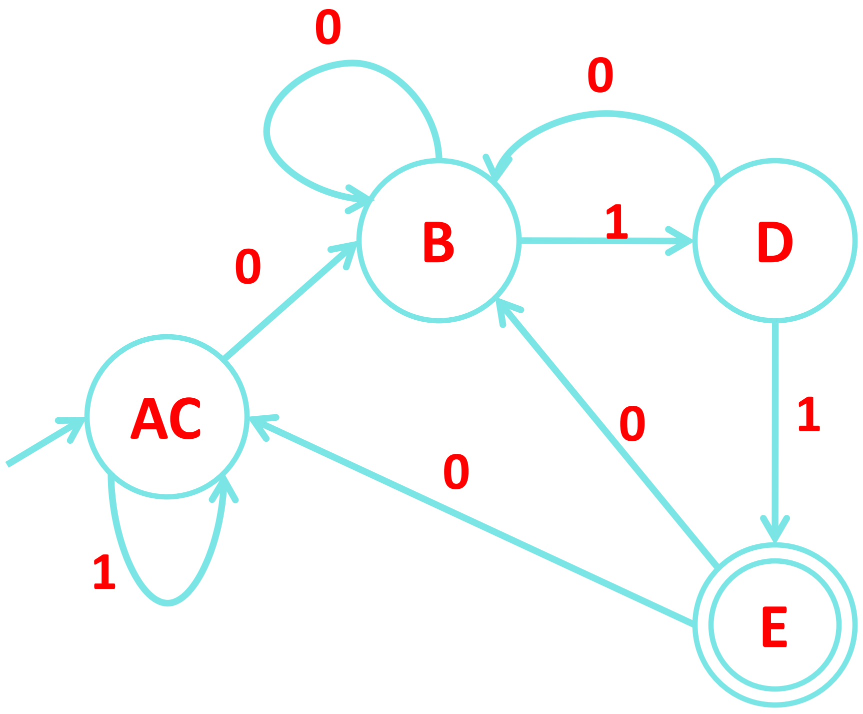
$\Pi_2 = \Pi_3$,so stop here.

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



Minimization of DFA

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].

Minimization of DFA

Using Table Filling Method (Myhill-Nerode Theorem)

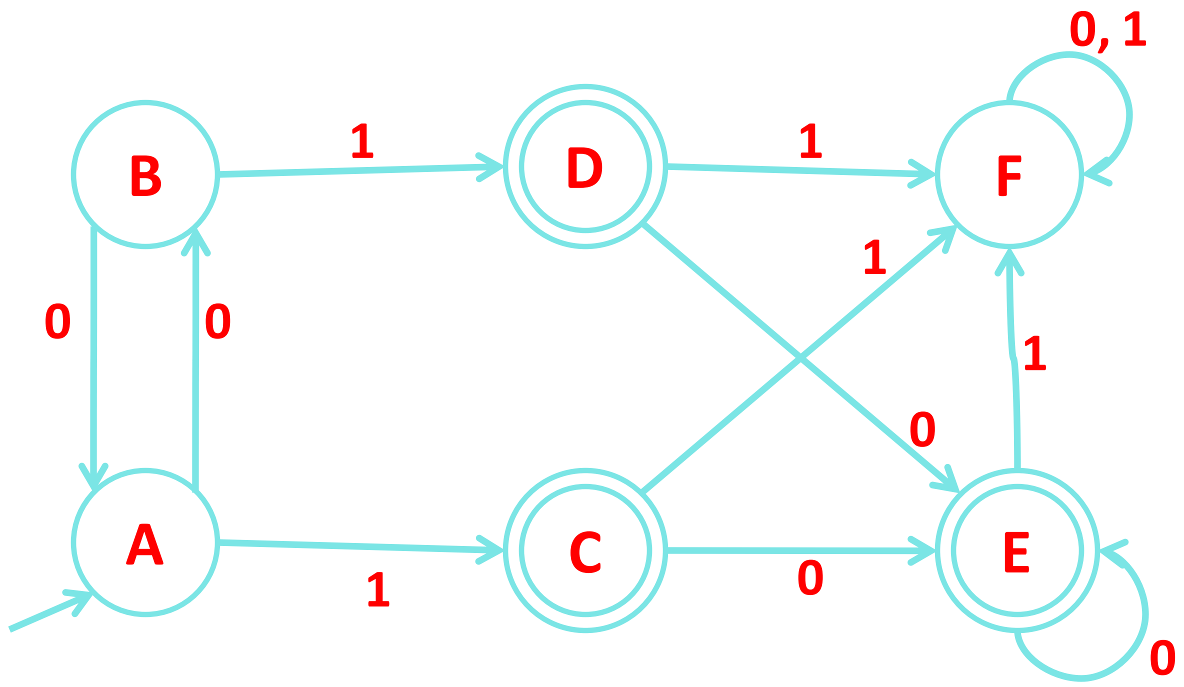
Contd…

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**	

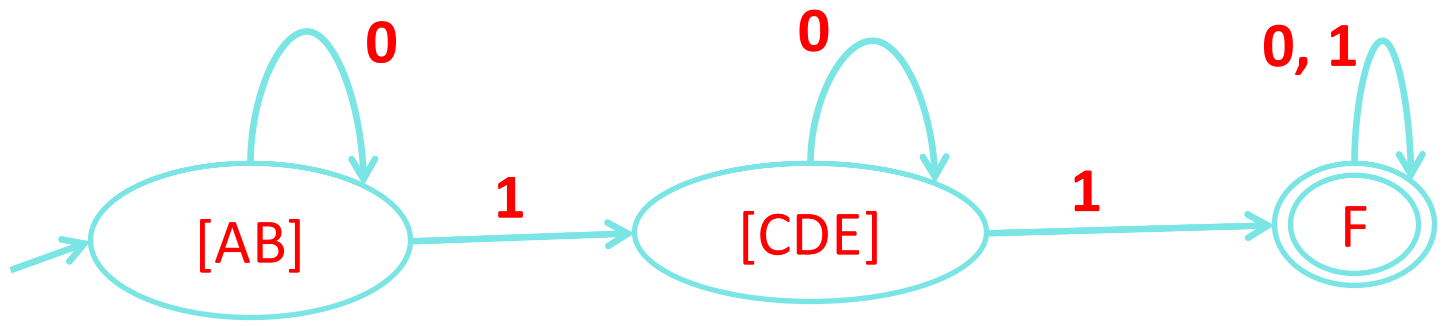
Minimization of DFA

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}.



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

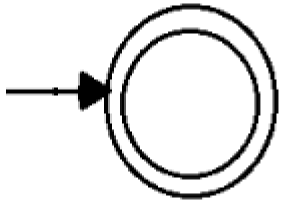
A. can

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

Finite Automata with Output

Objective of the Topic

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

- Understand the requirement of FA with output.
- Implement Mealy and Moore machine.
- Convert Mealy machine to Moore machine and vice-versa.

Finite Automata

Topic mapping with Course Outcome

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

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.

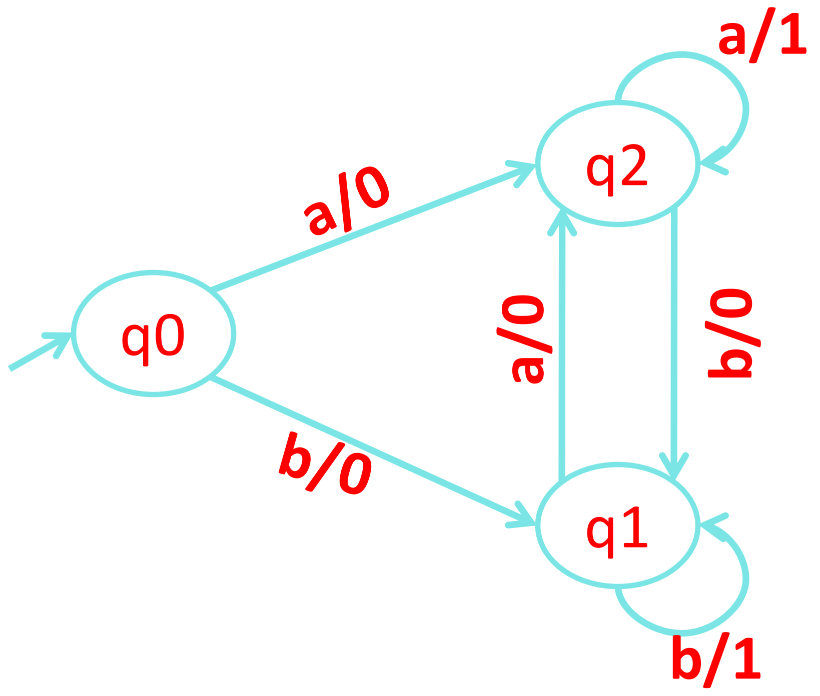
  δ = Transitional function mapping Q X Σ → Q

  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.

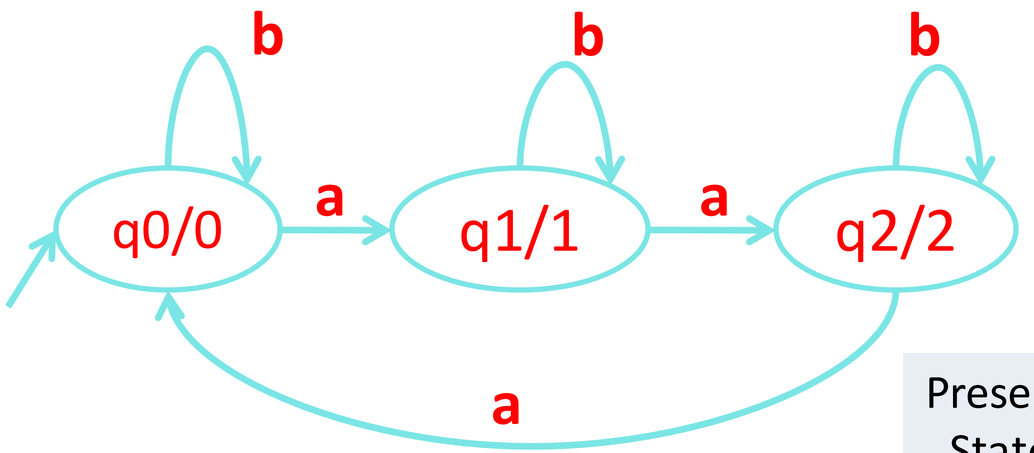
  δ = Transitional function mapping Q X Σ → Q

  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	q2	q1	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

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

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

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]**

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.

MCQ s

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

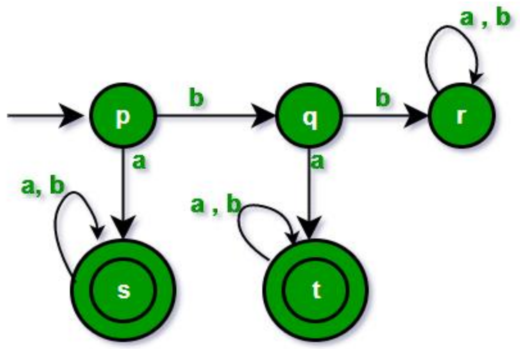
B. May be same

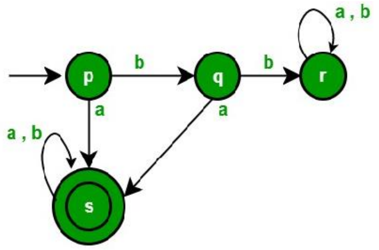
C. Must be different

D. None of the above

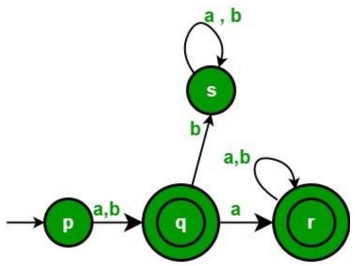
MCQ s

3. Which minimum state FA is equivalent to following FA

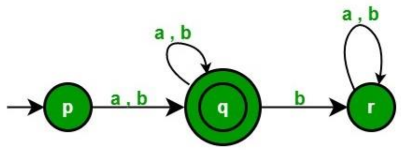




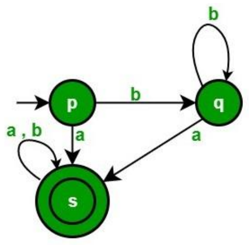
(A)



(B)



(C)



(D)

MCQ s

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}

MCQ s

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)

MCQ s

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 diferent 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

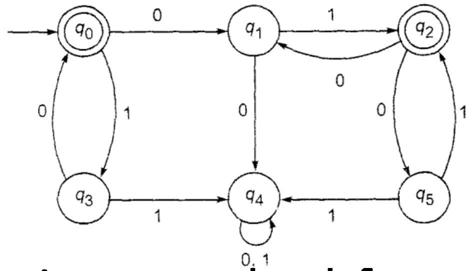
D. all of these

Old Question Papers

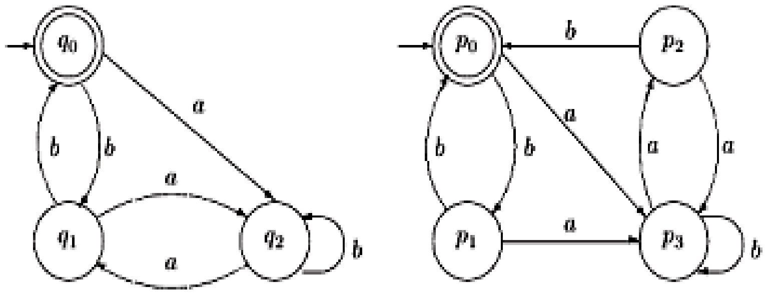
https://drive.google.com/drive/folders/10fjJwkU7_FW39oBResWiUJEHXddNQIPk?usp=sharing

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:



Summary

- 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, John E.; Motwani, Rajeev; Ullman, Jeffrey D. (2013). Introduction to Automata Theory, Languages, and Computation (3rd ed.). Pearson. ISBN 1292039051.*
- Peter Linz, "An Introduction to Formal Language and Automata", 4th Edition, Narosa Publishing house , 2006.
- John.C.martin, "Introduction to the Languages and the Theory of Computation",Third edition, Tata McGrawHill, 2003.

Thank You