

# Introduction to Theory of Computation: CS-202

# Syllabus

<b>CS 202</b>	<b>Theory of Computation</b>	<b>L</b>	<b>T</b>	<b>P</b>
		<b>3</b>	<b>0</b>	<b>0</b>

**Formal Language and Grammar:** Production systems, Chomsky Hierarchy, Right linear grammar and Finite stateautomata, Context free grammars, Normal forms, Derivation trees and ambiguity.

**Finite state Automata:** Non deterministic and deterministic FSA, NFSA with  $\epsilon$ - moves, RegularExpressions, Equivalence of regular expression and FSA, Pumping lemma, closure properties and decidability, Myhill - Nerode theorem and minimization, Finite automata with output.

**Pushdown automata:** Acceptance by empty store and final state, Equivalence between pushdownautomata and context-free grammars, Closure properties of CFL, Deterministic pushdownautomata.

**Turing Machines:** Techniques for Turing machine construction, Generalized and restricted versions equivalent to the basic model, Godel numbering, Universal Turing Machine, Recursivelyenumerable sets and recursive sets, Computable functions, time space complexity measures,context sensitive languages and linear bound automata.

**Decidability:** Post's correspondence problem, Rice's theorem, decidability of membership, emptiness and equivalence problems of languages.

## *Suggested Readings:*

1. J. E. Hopcraft, R. Motwani, J. D. Ullman, *Introduction to Automata Theory, Languages andComputation*, Pearson.
2. H. R. Lewis, C. H. Papadimitrou, *Elements of the Theory of Computation*, PHI.
3. P. Linz, *An Introduction to Formal Language and Automata*, Narosa Publisher.
4. K. L. P. Mishra, N. Chandrasekaran, *Theory of Computer Science: Automata, Languages andComputation*, PHI.

# Topic of course

- What are the fundamental capabilities and limitations of computers?
- To answer this, we will study abstract mathematical models of computers
- These mathematical models abstract away many of the details of computers to allow us to focus on the essential aspects of computation
- It allows us to develop a mathematical theory of computation

# Why study the theory of computing?

- Core mathematics of CS (has not changed in over 30 years)
- Many applications, especially in design of compilers and programming languages
- Important to be able to recognize uncomputable and intractable problems
- Need to know this in order to be a computer scientist, not simply a computer programmer

- Formal language
  - a subset of the set of all possible strings from a set of symbols
  - example: the set of all syntactically correct C programs
- Automata
  - abstract, mathematical model of computer
  - examples: finite automata, pushdown automata Turing machine, RAM, PRAM, many others
- Let's consider each of these in turn

# Formal language

**Alphabet** = finite set of symbols or characters

examples:  $\Sigma = \{a,b\}$ , binary, ASCII

**String** = finite sequence of symbols from an alphabet

examples: aab, bbaba, also computer programs

A **formal language** is a set of strings over an alphabet

Examples of formal languages over alphabet  $\Sigma = \{a, b\}$ :

$$L_1 = \{aa, aba, aababa, aa\}$$

$$L_2 = \{\text{all strings containing just two a's and any number of b's}\}$$

A formal language can be finite or infinite.

# Formal languages (continued)

We often use **string variables** ;  $u = aab$ ,  $v = bbaba$

Operations on strings

length:  $|u| = 3$

reversal:  $u^R = baa$

concatenation:  $uv = aabbaba$

The **empty string** , denoted  $\lambda$  , has some special properties:

$$|\lambda| = 0$$

$$\lambda w = w \lambda = w$$

# Formal languages (continued)

If  $w$  is a string, then  $w^n$  stands for the string obtained by repeating  $w$   $n$  times.

$$w^0 = \lambda$$

$$\Sigma^+ = \Sigma^* - \{\lambda\}$$

$$L^0 = \{\lambda\}$$

$$L^1 = L$$

## Important example of a formal language

- alphabet: ASCII symbols
- string: a particular C++ program
- formal language: set of all legal C++ programs

# Grammars

A grammar  $G$  is defined as a quadruple:

$$G = (V, T, S, P)$$

Where  $V$  is a finite set of objects called variables

$T$  is a finite set of objects called terminal symbols

$S \in V$  is a special symbol called the Start symbol

$P$  is a finite set of productions or "production rules"

Sets  $V$  and  $T$  are nonempty and disjoint

# Grammars

What is the relationship between a language and a grammar?

Let  $G = (V, T, S, P)$

The set

$$L(G) = \{w \in T^* : S \xrightarrow{*} w\}$$

is the language generated by G.

# Grammars

Consider the grammar  $G = (V, T, S, P)$ , where:

$$V = \{S\}$$

$$T = \{a, b\}$$

$$S = S,$$

$$P = \boxed{S \rightarrow aSb}$$

$$\boxed{S \rightarrow \lambda}$$

# Grammars

What are some of the strings in this language?

$S \Rightarrow aSb \Rightarrow ab$

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$

It is easy to see that the language generated by this grammar is:

$$L(G) = \{a^n b^n : n \geq 0\}$$

(See proof on pp. 21-22 in Linz)

# Language-recognition problem

- There are many types of computational problem. We will focus on the simplest, called the “language-recognition problem.”
- Given a string, determine whether it belongs to a language or not. (Practical application for compilers: Is this a valid C++ program?)
- We study simple models of computation called “automata,” and measure their computational power in terms of the class of languages they can recognize.

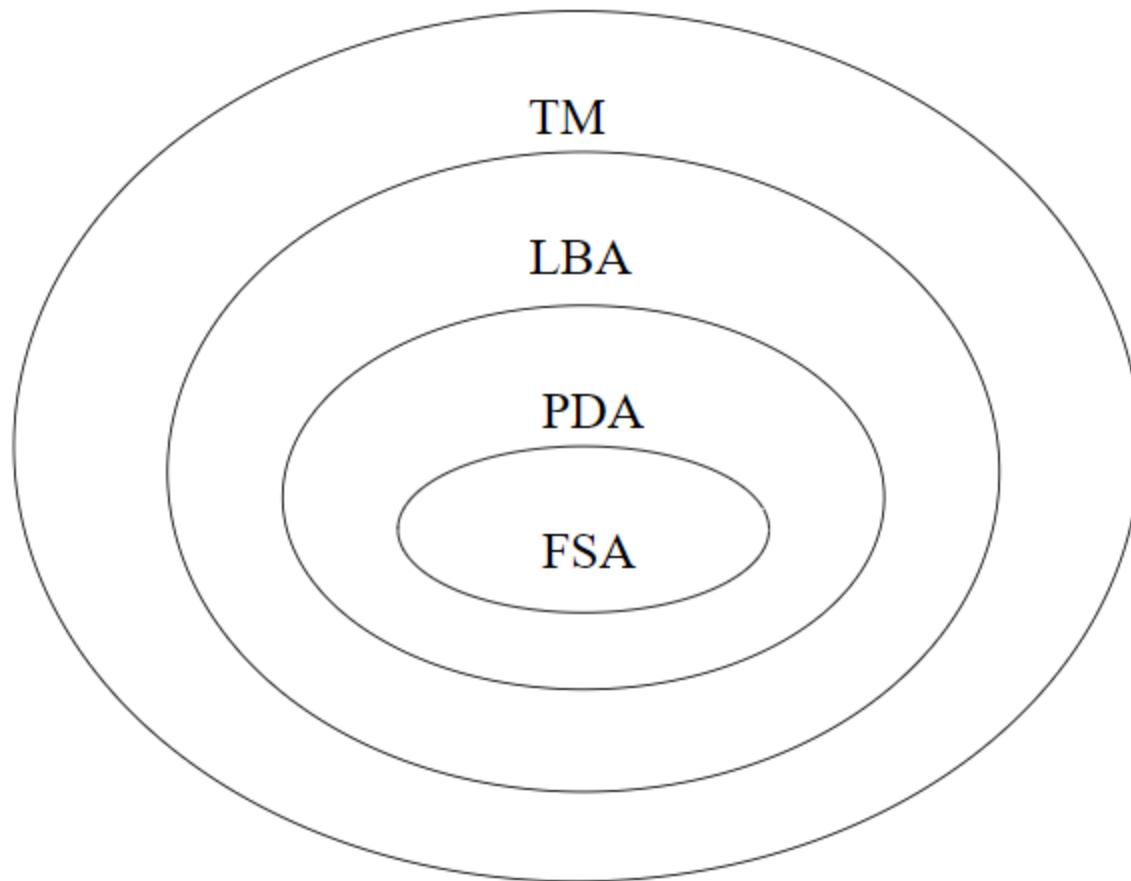
# Finite automata

- Developed in 1940's and 1950's for neural net models of brain and computer hardware design
- *Finite memory!*
- Many applications:
  - text-editing software: search and replace
  - many forms of pattern-recognition (including use in WWW search engines)
  - compilers: recognizing keywords (lexical analysis)
  - sequential circuit design
  - software specification and design
  - communications protocols

# Pushdown automata

- Noam Chomsky's work in 1950's and 1960's on grammars for natural languages
- infinite memory, organized as a stack
- Applications:
  - compilers: parsing computer programs
  - programming language design

# Computational power



# Automata, languages, and grammars

- In this course, we will study the relationship between automata, languages, and grammars
- Recall that a formal language is a set of strings over a finite alphabet
- Automata are used to *recognize* languages
- Grammars are used to *generate* languages
- (All these concepts fit together)

# Classification of automata, languages, and grammars

Automata	Language	Grammar
Turing machine	Recursively enumerable	Recursively enumerable
Linear-bounded automaton	Context sensitive	Context sensitive
Nondeterministic push-down automaton	Context free	Context free
Finite-state automaton	regular	regular

Besides developing a theory of classes of languages and automata, we will study the limits of computation. We will consider the following two important questions:

- What problems are impossible for a computer to solve?
- What problems are too difficult for a computer to solve in practice (although possible to solve in principle)?

# Uncomputable (undecidable) problems

- Many well-defined (and apparently simple) problems cannot be solved by any computer
- Examples:
  - For any program  $x$ , does  $x$  have an infinite loop?
  - For any two programs  $x$  and  $y$ , do these two programs have the same input/output behavior?
  - For any program  $x$ , does  $x$  meet its specification?  
(i.e., does it have any bugs?)

**Thank you**

# Theory of Computation: CS-202

# Content

- Review of set theory
- Formal Language
- String Operations
- Grammar

# Review of set theory

A Set is a well defined collection of distinct objects.

- list of elements:  $A = \{6, 12, 28\}$
- characteristic property:  $B = \{x \mid x \text{ is a positive, even integer}\}$

Set membership:  $12 \in A$ ,  $9 \notin A$

# Barber Paradox

Barber: One who shaves all those and those only, who do not shave themselves.

Question: Does Barber shave himself?

1. If he shaves himself he ceases to be Barber.  
(As Barber cant shave himself.)                      Contradiction
2. If the Barber does not shave himself then he must shave himself.  
(Barber can shave himself)                              Contradiction

- Subset- If A and B are two set, such that every element of A is also an element of B then  $A \subseteq B$ .
- Proper Subset- If A is subset of B, but B contains an element not in A, then  $A \subset B$ .
  - list of elements:  $A = \{6, 12, 28\}$
  - characteristic property:  $B = \{x \mid x \text{ is a positive, even integer}\}$

Set inclusion:  
 $A \subseteq B$  (A is a subset of B)  
 $A \subset B$  (A is a proper subset of B)

- Disjoin set- If A and B has no common element then  $A \cap B = \emptyset$ , then the sets are said to be disjoint.
- Null Set-A set with no element is called null set.

Eg. {} or  $\emptyset$

## Set Operations

<b>Union (<math>\cup</math>)</b>	$: S_1 \cup S_2 = \{x : x \in S_1 \text{ or } x \in S_2\}$
<b>Intersection (<math>\cap</math>)</b>	$: S_1 \cap S_2 = \{x : x \in S_1 \text{ and } x \in S_2\}$
<b>Difference (-)</b>	$: S_1 - S_2 = \{x : x \in S_1 \text{ and } x \notin S_2\}$
<b>Complement</b>	$: \bar{S} = \{x : x \in U \text{ and } x \notin S\}$

Example: Consider the set A and B

$$A = \{6, 12, 28\}$$

$$B = \{x \mid x \text{ is a positive, even integer}\}$$

then

$$A \cup \emptyset = A$$

$$A \cap \emptyset = A$$

$$\emptyset' = U$$

union:  $A \cup \{9, 12\} = \{6, 9, 12, 28\}$

intersection:  $A \cap \{9, 12\} = \{12\}$

difference:  $A - \{9, 12\} = \{6, 28\}$

# Set theory (continued)

Another set operation, called “taking the complement of a set”, assumes a **universal set** .

Let  $U = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$  be the universal set.

Let  $A = \{2, 4, 6, 8\}$

Then  $\bar{A} = U - A = \{0, 1, 3, 5, 7, 9\}$

The **empty set** :  $\emptyset = \{\}$

## Set theory (continued)

The **cardinality** of a set is the number of elements in a set.

Let  $S = \{2, 4, 6\}$

Then  $|S| = 3$

The **powerset** of  $S$ , represented by  $2^S$ , is the set of all subsets of  $S$ .

$2^S = \{\{\}, \{2\}, \{4\}, \{6\}, \{2, 4\}, \{2, 6\}, \{4, 6\}, \{2, 4, 6\}\}$

The number of elements in a powerset is  $|2^S| = 2^{|S|}$

# Theory of Computation

## Basic Concepts

- *Automaton*: a formal construct that accepts input, produces output, may have some temporary storage, and can make decisions
- *Formal Language*: a set of sentences formed from a set of symbols according to formal rules
- *Grammar*: a set of rules for generating the sentences in a formal language

In addition, the theory of computation is concerned with questions of computability (the types of problems computers can solve in principle) and complexity (the types of problems that can be solved in practice).

# Formal language

**Alphabet** = finite set of symbols or characters

examples:  $\Sigma = \{a,b\}$ , binary, ASCII

**String** = finite sequence of symbols from an alphabet

examples: aab, bbaba, also computer programs

A **formal language** is a set of strings over an alphabet

Examples of formal languages over alphabet  $\Sigma = \{a, b\}$ :

$$L_1 = \{aa, aba, aababa, aa\}$$

$$L_2 = \{\text{all strings containing just two a's and any number of b's}\}$$

A formal language can be finite or infinite.

# String Operations

**String Concatenation:** The concatenation of two strings  $w$  and  $v$  is the string obtained by appending the symbols of  $v$  to the right of  $w$ .

Eg.       $w=a_1a_2\dots a_n$   
               $v=b_1b_2\dots b_n$   
               $wv=a_1a_2\dots a_n b_1b_2\dots b_n$

**String Reversal:** It is obtained by writing the symbols in reverse order.

Eg.       $w=a_1a_2\dots a_n$   
               $w^R=a_na_{n-1}\dots a_1$

**String Length:** Let  $w$  be a string then  $|w|$  is the number of symbols in the string.

$w=a_1a_2$   
 $|w|=2$

**Sub-String:** Any string of consecutive character in some w is called Sub-String.  
Example,

$$w = a_1 a_2 a_3 a_4 a_5$$

Substring of w =  $a_1 a_2, a_2 a_3, \dots$  etc

**Prefix and suffix:** Let  $w = vu$  be a string then the substring v and u are said to be a prefix and suffix of w.

Example,

$$w = abbab$$

Then, prefix = { $\lambda, a, ab, abb, abba, abbab$ }

$$\text{Sufix} = \{\lambda, b, ab, bab, bbab\}$$

The **empty string**, denoted  $\lambda$ , has some special properties:

$$|\lambda| = 0$$

$$\lambda w = w \lambda = w$$

# Kleen or Star Closure $\Sigma^*$

If  $w$  is a string, then  $w^n$  stands for the string obtained by repeating  $w n$  times.

$$w^0 = \lambda$$

$\Sigma^*$ =set of string obtained by concatenating zero or more symbols from  $\Sigma$ .

If  $\Sigma=\{a,b\}$

$$\Sigma^0=\{\lambda\}$$

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

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

.....

$$\Sigma^* = \{ \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots \}$$

$$\Sigma^+ = \Sigma^* - \{\lambda\}$$

Note:  $\Sigma$  is finite by assumption,  $\Sigma^*$  and  $\Sigma^+$  will always be infinite.

# Language

- It is defined as a subset of  $\Sigma^*$ . Any set of strings on the alphabet  $\Sigma$  can be considered as a language

Example 1:  $\Sigma = \{a, b\}$

$$\Sigma^* = \{\lambda, a, b, ab, ba, \dots\}$$

Then the set  $S = \{a, aa, aab\}$  is a language on  $\Sigma$ .

Example 2:  $L = \{a^n b^n : n \geq 0\}$  is also a language on  $\Sigma$ .

The strings  $ab, aabb, aaabbb \in L$

While  $abb \notin L$ .

# Language (Continue...)

Note: Since languages are sets, so the union, intersection and difference of two languages are immediately defined.

# Operations on languages

Set operations:

$L_1 \cup L_2 = \{x \mid x \in L_1 \text{ or } x \in L_2\}$  is union

$L_1 \cap L_2 = \{x \mid x \in L_1 \text{ and } x \in L_2\}$  is intersection

$L_1 - L_2 = \{x \mid x \in L_1 \text{ and } x \notin L_2\}$  is difference

$\bar{L} = \Sigma^* - L$  is complement

$L_1 \oplus L_2 = (L_1 - L_2) \cup (L_2 - L_1)$  is “symmetric difference”

String operations:

$L^R = \{w^R \mid w \in L\}$  is “reverse of language”

$L_1 L_2 = \{xy \mid x \in L_1, y \in L_2\}$  is “concatenation of languages”

$L^* = L^0 \cup L^1 \cup L^2 \cup \dots$  is “Kleene star” or “star closure”

$L^+ = L^1 \cup L^2 \cup \dots$  is positive closure

# Grammars

A grammar  $G$  is defined as a quadruple:

$$G = (V, T, S, P)$$

Where  $V$  is a finite set of objects called variables

$T$  is a finite set of objects called terminal symbols

$S \in V$  is a special symbol called the Start symbol

$P$  is a finite set of productions or "production rules"

Sets  $V$  and  $T$  are nonempty and disjoint

All production rules are of the form

$X \rightarrow Y$

Where,  $X \in (V \cup T)^+$

$Y \in (V \cup T)^*$

& the productions are applied as follows:

Suppose

$w = uXv,$

And  $X \rightarrow y$

$\Rightarrow z = uyv$

$w \Rightarrow z$

We say that  $w$  derives  $z$  or  $z$  is derived from  $w$ .

A production can be used whenever it is applicable, if

$$w_1 \Rightarrow w_2 \Rightarrow w_3 \Rightarrow w_4 \Rightarrow \dots \Rightarrow w_n$$

we say that  $\Rightarrow w_1 \xrightarrow{*} w_n$

Here, \* indicates that an unspecified number of steps (including zero) can be taken to derive  $w_n$  from  $w_1$ .

Note: By applying the production rules in a different order, a given grammar can normally generate many strings. The set of all such strings is the language defined or generated by the grammar.

# Grammars

What is the relationship between a language and a grammar?

Let  $G = (V, T, S, P)$

The set

$$L(G) = \{w \in T^* : S \xrightarrow{*} w\}$$

is the language generated by G.

If  $w \in L(G)$  then the sequence

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow w_3 \Rightarrow w_4 \Rightarrow \dots \Rightarrow w_n \Rightarrow w$$

is a derivation of the sentence  $w$ .

The strings  $S, w_1, w_2, \dots, w_n$  which contain variable as well as terminals are called “sentential form” of the derivation and ‘ $w$ ’ which contain all terminal is called sentence.

Example: 1

Consider the grammar

$$G = (\{S\}, \{a, b\}, S, P)$$

With P is given by

$$1. S \rightarrow aSb$$

$$2. S \rightarrow \lambda$$

$$\begin{array}{ccc} 1 & 1 & 2 \end{array}$$

Then  $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$

So we can write

$$S \Rightarrow aabb$$

String aabb is a sentence and aaSbb is a sentential form.

$$\text{So, } L(G) = \{a^n b^n : n \geq 0\}$$

## Example: 2

Consider the grammar

$$G_1 = (\{S, A\}, \{a, b\}, S, P_1)$$

With  $P_1$  given by

1.  $S \rightarrow aAb$
2.  $S \rightarrow \lambda$
3.  $A \rightarrow aAb$
4.  $A \rightarrow \lambda$

1            3            4

Then  $S \Rightarrow aAb \Rightarrow aaAbb \Rightarrow aabb$

So we can write

$$S \Rightarrow aabb$$

String aabb is a sentence and aaAbb is a sentential form.

$$\text{So, } L(G_1) = \{a^n b^n : n \geq 0\}$$

# Equivalence of grammar

- Two grammars are equivalent if they generate the same language.
- In the above example  $G$  and  $G_1$  are equivalent

## Example: 3

Find the grammar that generates language

$$L = \{a^n b^{n+1} : n \geq 0\}$$

Let  $G = (\{S, A\}, \{a, b\}, S, P)$

With P:

1.  $S \rightarrow Ab$
2.  $A \rightarrow aAb$
3.  $A \rightarrow \lambda$

## Example: 4

Take  $\Sigma = \{a, b\}$  and let  $n_a(w)$  and  $n_b(w)$  denotes the number of a's and b's in the string w the G with production

1.  $S \rightarrow SS$
2.  $S \rightarrow \lambda$
3.  $S \rightarrow aSb$
4.  $S \rightarrow bSa$

Generate  $L(G)$ .

$$L = \{w : n_a(w) = n_b(w)\}$$

## Suggested readings

1. An introduction to FORMAL LANGUAGES and AUTOMATA by PETER LINZ.
2. Introduction to Automata Theory, Languages, And Computation by JOHN E. HOPCROFT, RAJEEV MOTWANI, JEFFREY D. ULLMAN
3. Theory of computer science: automata, languages and computation by K.L.P MISHRA

**Thank you**

# Theory of Computation: CS-202

## Deterministic Finite Automata

# Content

## Finite Automata

### Deterministic Finite Accepters

- Deterministic Accepters
- Transition Graphs
- Languages and DFAs
- Regular Language

# Finite Automata

1. Deterministic Finite Automata
2. Non-Deterministic Finite Automata

# Deterministic Finite Accepters

## Definition

A **deterministic finite accepter** or **dfa** is defined by the quintuple

$$M = (Q, \Sigma, \delta, q_0, F)$$

where  $Q$  is a finite set of **internal states**,

$\Sigma$  is a finite set of symbols called the **input alphabet**,

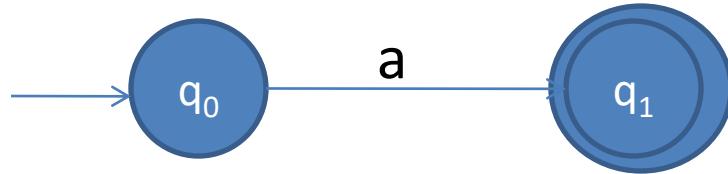
$\delta: Q \times \Sigma \rightarrow Q$  is a **total** function called the **transition function**,

$q_0 \in Q$  is the **initial state**,

$F \subseteq Q$  is a set of **final states**.

# Transition Graph

- To visualize and represent finite automata, we use transition graphs, in which vertices represent states and the edges represent transitions.
- The labels on the vertices are the name of the states, while the labels on the edges are the current values of the input symbol.



- The initial state is identified by an incoming unlabeled arrow not originated at any vertex.
- Final states are drawn with a double circle.

# Example

Transition Graph of a dfa  $M = (Q, \Sigma, \delta, q_0, F)$

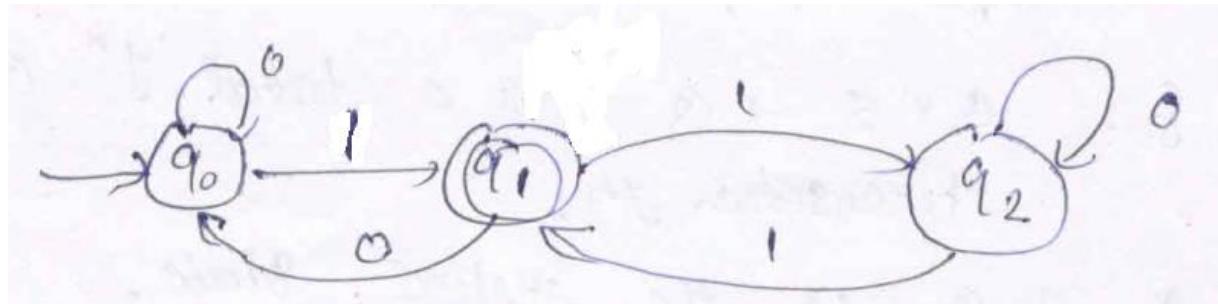
Vertex labeled with  $q_i$ : state  $q_i \in Q$ ,

Edge from  $q_i$  to  $q_j$  labeled with  $a$ : transition  $\delta(q_i, a) = q_j$ .

Example .1  $M = (\{q_0, q_1, q_3\}, \{0,1\}, \delta, q_0, \{q_1\})$ , where  $\delta$  is given by

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

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



**Transition Graph** of a dfa  $M = (Q, \Sigma, \delta, q_0, F)$

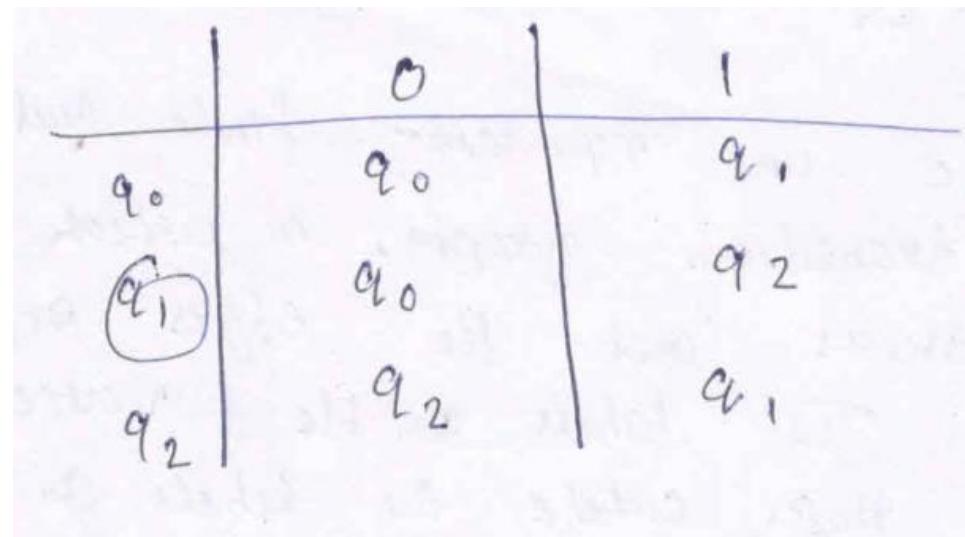
Vertex labeled with  $q_i$ : state  $q_i \in Q$ ,

Edge from  $q_i$  to  $q_j$  labeled with  $a$ : transition  $\delta(q_i, a) = q_j$ .

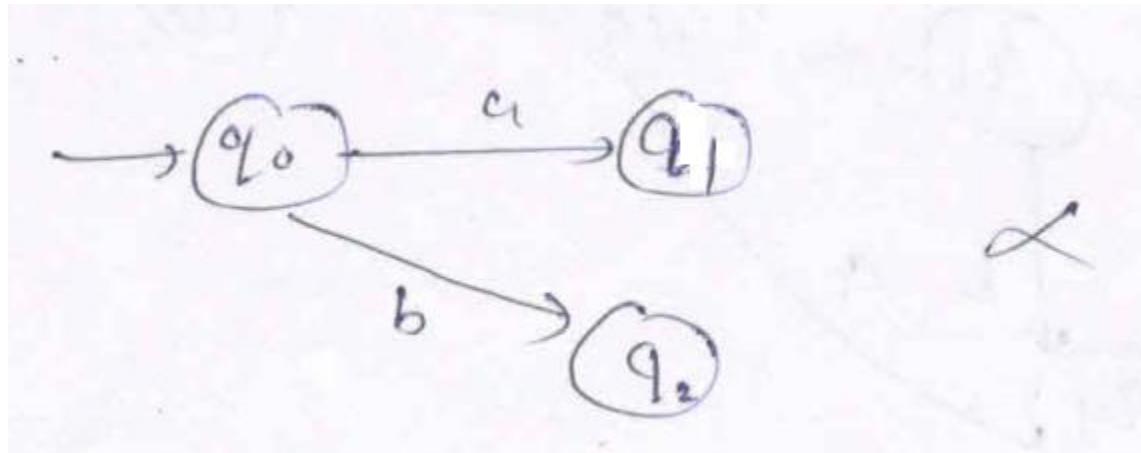
Example .1  $M = (\{q_0, q_1, q_3\}, \{0,1\}, \delta, q_0, \{q_1\})$ , where  $\delta$  is given by

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

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

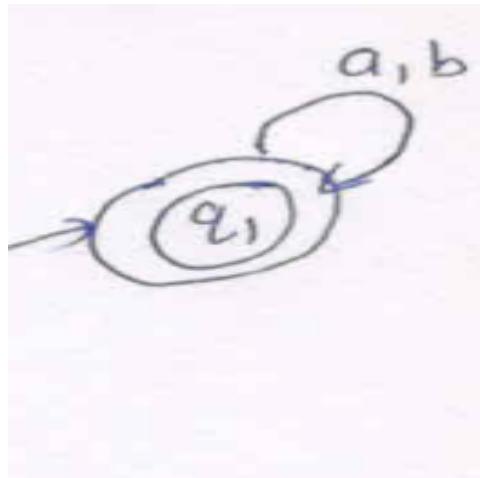


- Note: Machine should be complete

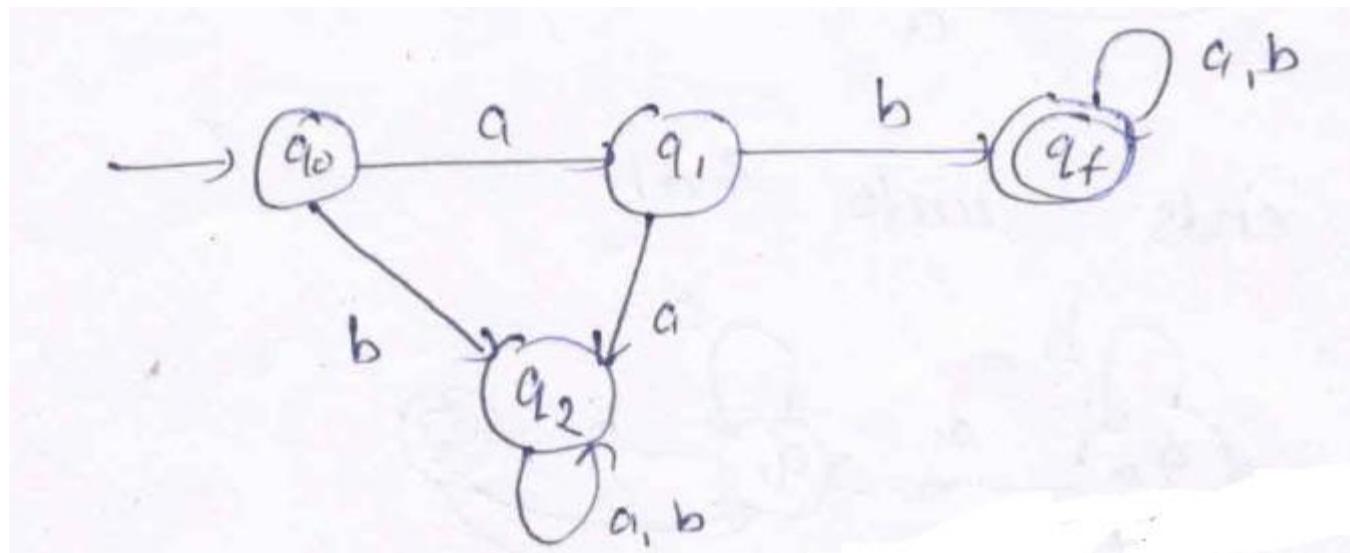


- The processed symbol is remembered by changing the state.

# Number of final states



- Draw a DFA which accepts all the strings on  $\Sigma = \{a, b\}$  with the prefix 'ab'.



## Suggested readings

1. An introduction to FORMAL LANGUAGES and AUTOMATA by PETER LINZ.
2. Introduction to Automata Theory, Languages, And Computation by JOHN E. HOPCROFT, RAJEEV MOTWANI, JEFFREY D. ULLMAN
3. Theory of computer science: automata, languages and computation by K.L.P MISHRA

**Thank you**

# Theory of Computation: CS-202

## Deterministic Finite Automata-2

# Content

## Finite Automata

### Deterministic Finite Accepters

- Deterministic Accepters
- Transition Graphs
- Examples

# Deterministic Finite Accepters

## Definition

A **deterministic finite accepter** or **dfa** is defined by the quintuple

$$M = (Q, \Sigma, \delta, q_0, F)$$

where  $Q$  is a finite set of **internal states**,

$\Sigma$  is a finite set of symbols called the **input alphabet**,

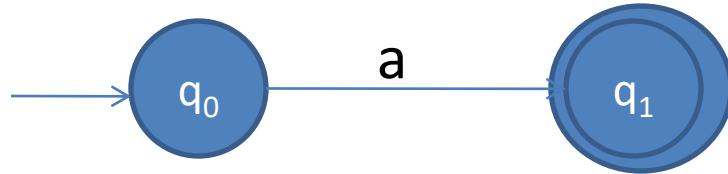
$\delta: Q \times \Sigma \rightarrow Q$  is a **total** function called the **transition function**,

$q_0 \in Q$  is the **initial state**,

$F \subseteq Q$  is a set of **final states**.

# Transition Graph

- To visualize and represent finite automata, we use transition graphs, in which vertices represents states and the edges represents transitions.
- The labels on the vertices are the name of the states, while the labels on the edges are the current values of the input symbol.



- The initial state is identified by an incoming unlabeled arrow not originated at any vertex.
- Final states are drawn with a double circle.

# Example

Transition Graph of a dfa  $M = (Q, \Sigma, \delta, q_0, F)$

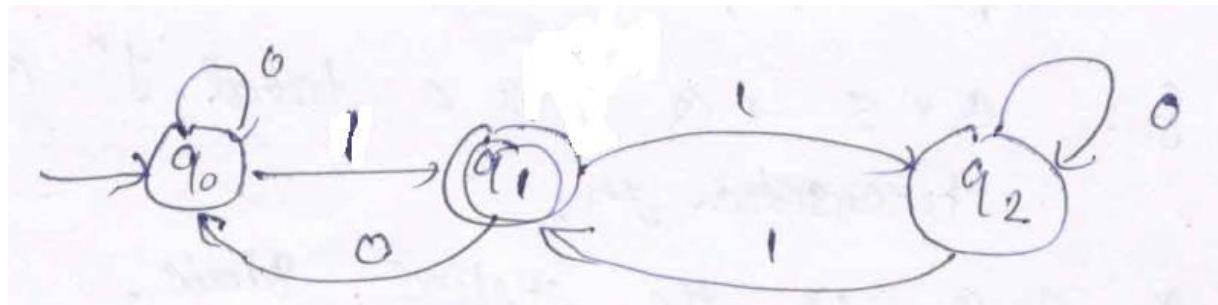
Vertex labeled with  $q_i$ : state  $q_i \in Q$ ,

Edge from  $q_i$  to  $q_j$  labeled with  $a$ : transition  $\delta(q_i, a) = q_j$ .

Example .1  $M = (\{q_0, q_1, q_3\}, \{0,1\}, \delta, q_0, \{q_1\})$ , where  $\delta$  is given by

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

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



**Transition Graph** of a dfa  $M = (Q, \Sigma, \delta, q_0, F)$

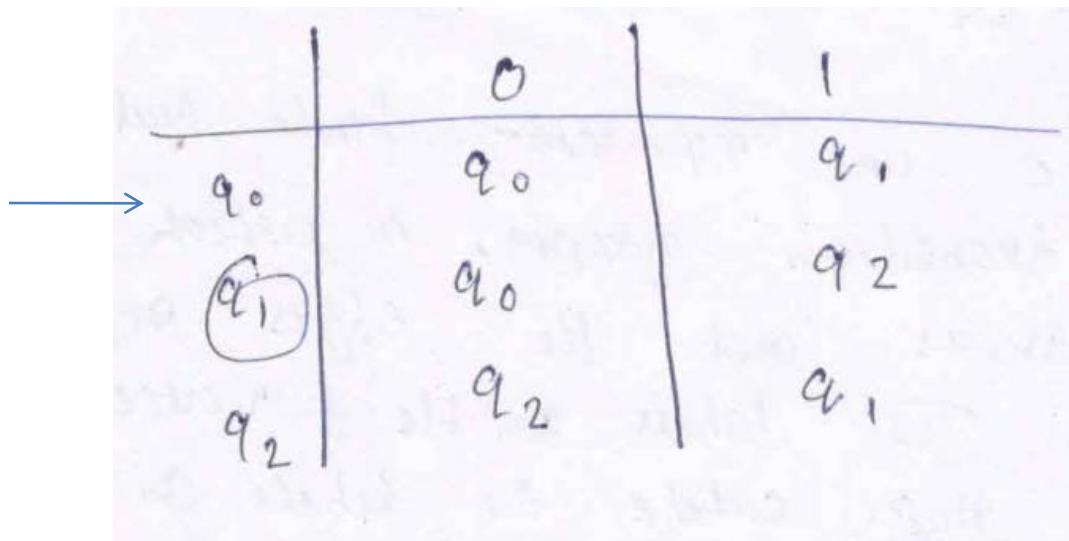
Vertex labeled with  $q_i$ : state  $q_i \in Q$ ,

Edge from  $q_i$  to  $q_j$  labeled with  $a$ : transition  $\delta(q_i, a) = q_j$ .

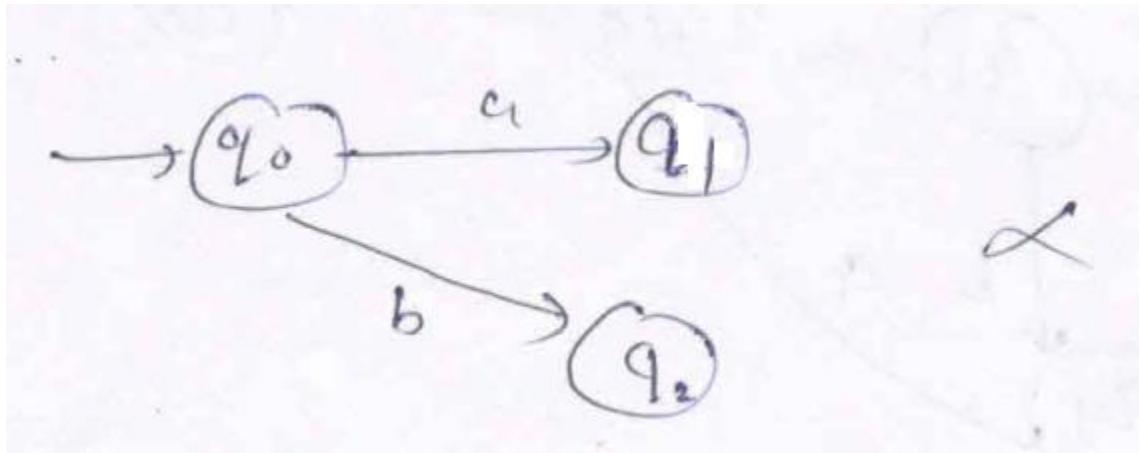
Example .1  $M = (\{q_0, q_1, q_3\}, \{0,1\}, \delta, q_0, \{q_1\})$ , where  $\delta$  is given by

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

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

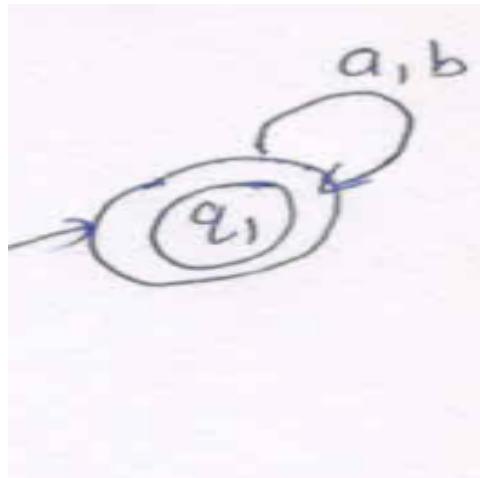


- Note: Machine should be complete



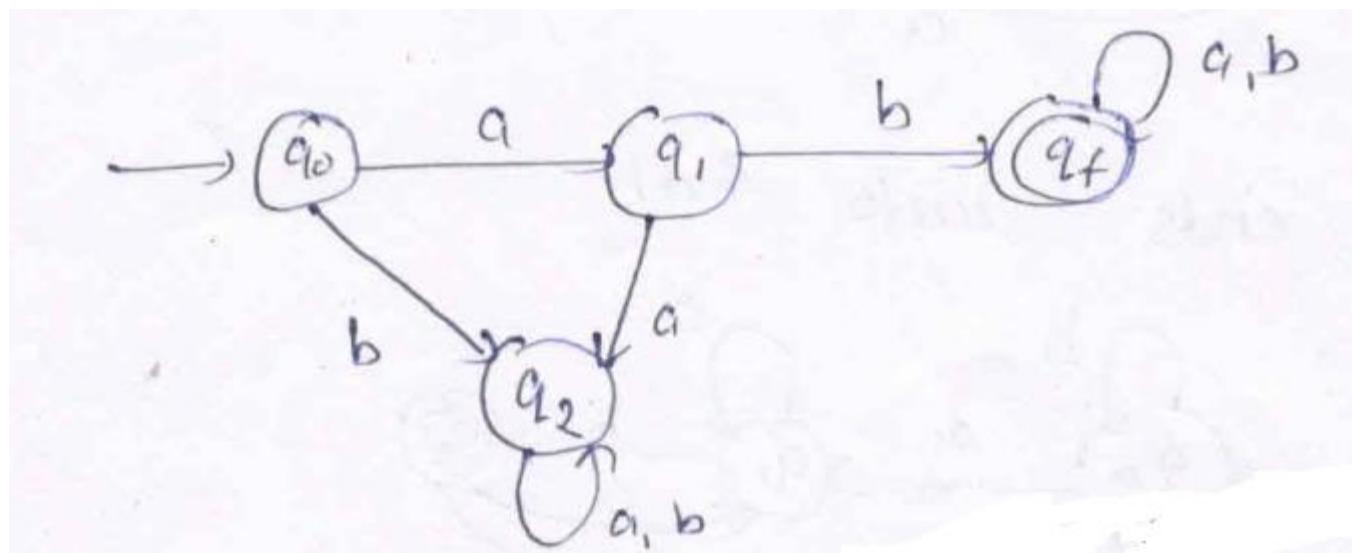
- The processed symbol is remembered by changing the state.

# Number of final states



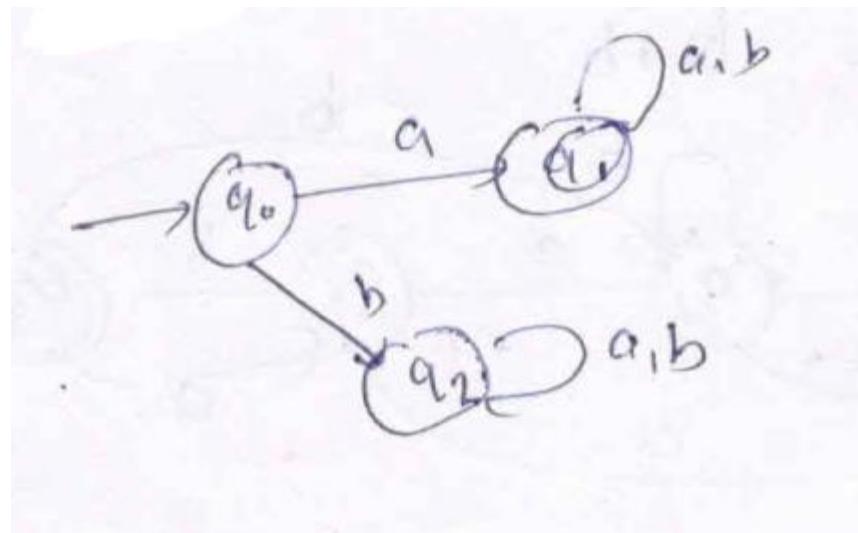
# Examples

1. Draw a DFA which accepts all the strings on  $\Sigma=\{a,b\}$  with the prefix 'ab'.



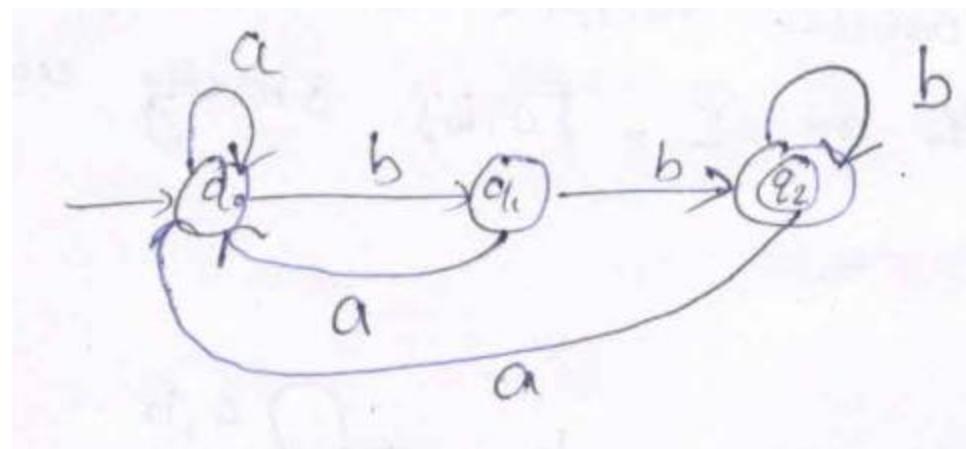
# Example (cont..)

2. Draw a DFA which accepts all the strings on  $\Sigma=\{a,b\}$  which starts with 'a'.



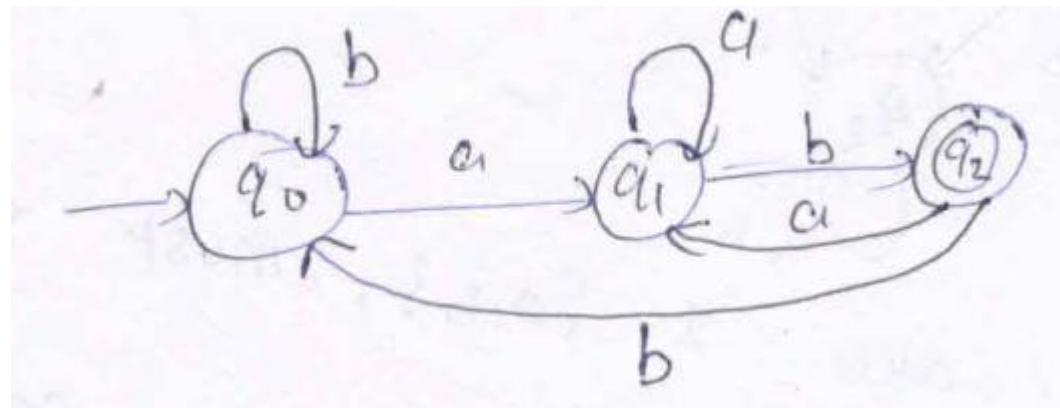
3. Draw a DFA which accepts all the strings on  $\Sigma=\{a,b\}$  which must end with ‘bb’.

Input string	b		a		a		b		b		b
--------------	---	--	---	--	---	--	---	--	---	--	---



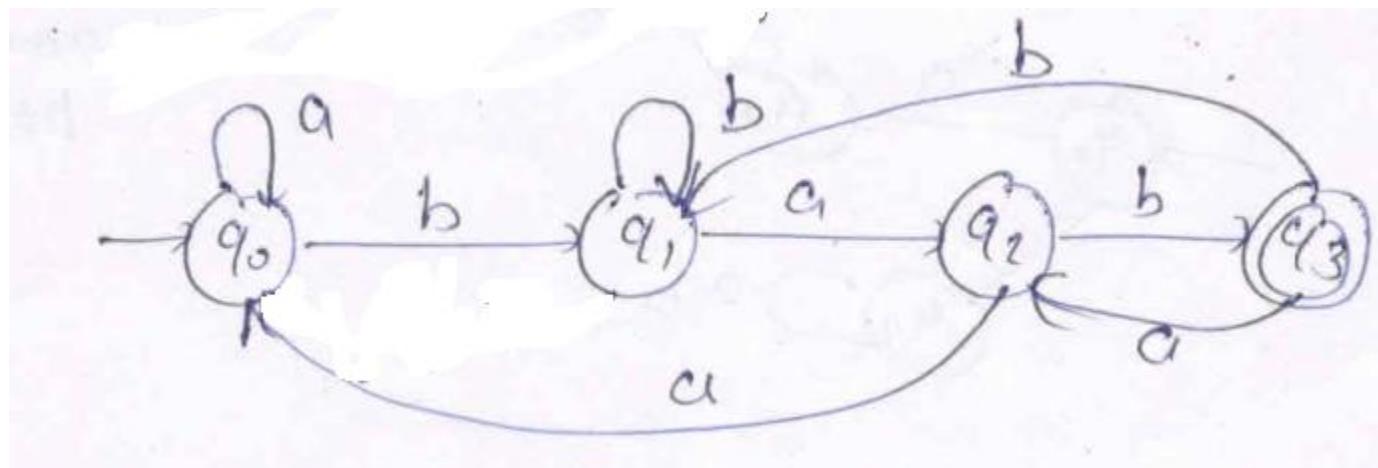
4. Draw a DFA which accepts all the strings on  $\Sigma=\{a,b\}$  which must end with 'ab'.

Input string      b | a | a | b | a | b

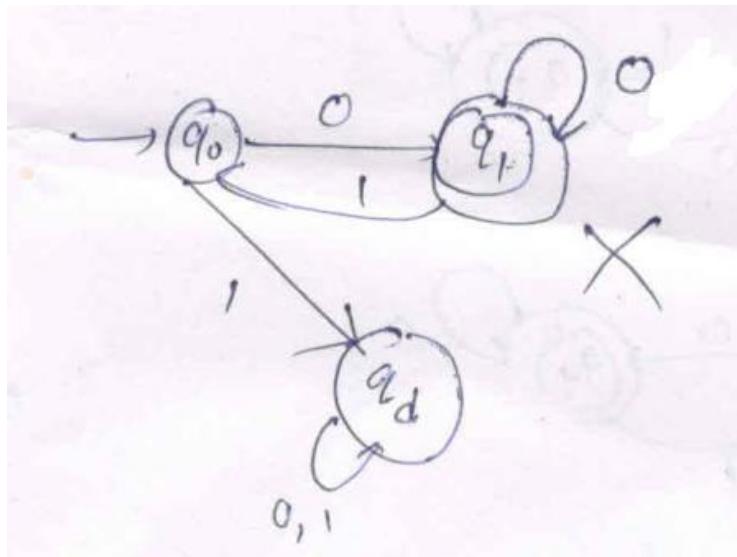


5. Draw a DFA which accepts all the strings on  $\Sigma=\{a,b\}$  which must end with ‘bab’.

Input string      b            |      a            |      a            |      b            |      a            |      b



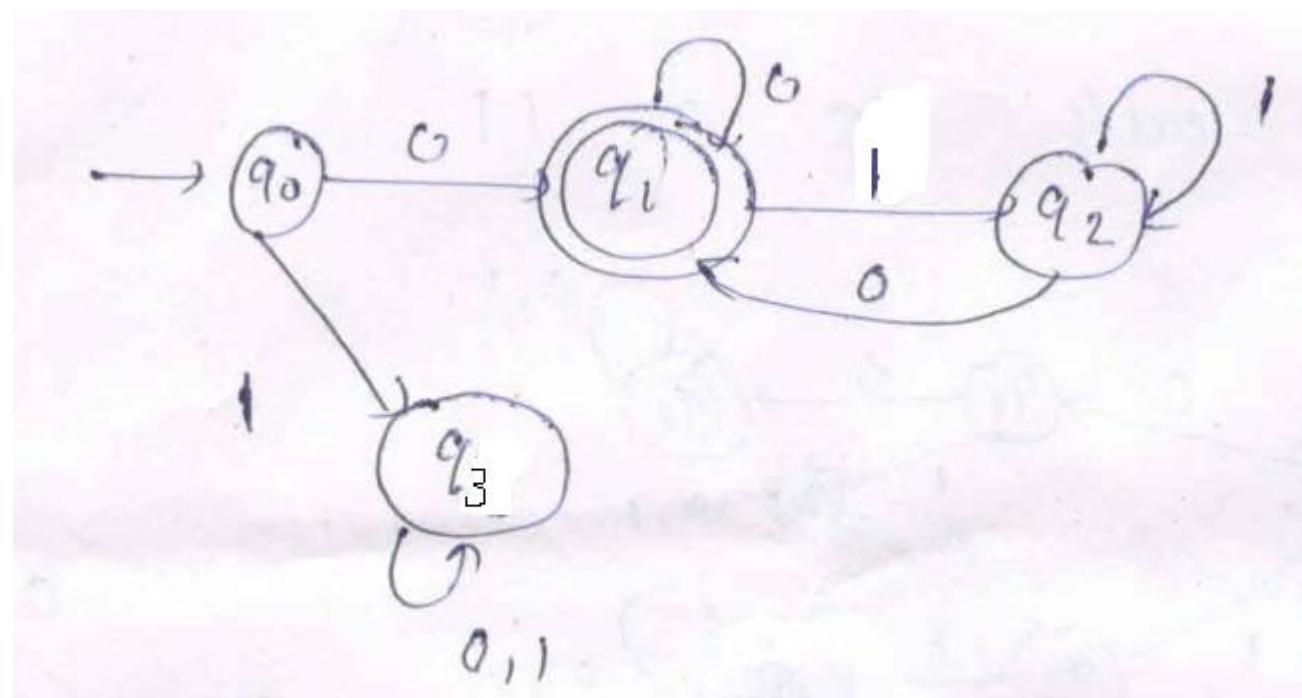
6. Draw a DFA which accepts all the strings on  $\Sigma=\{0,1\}$  which must start & end with '0'.



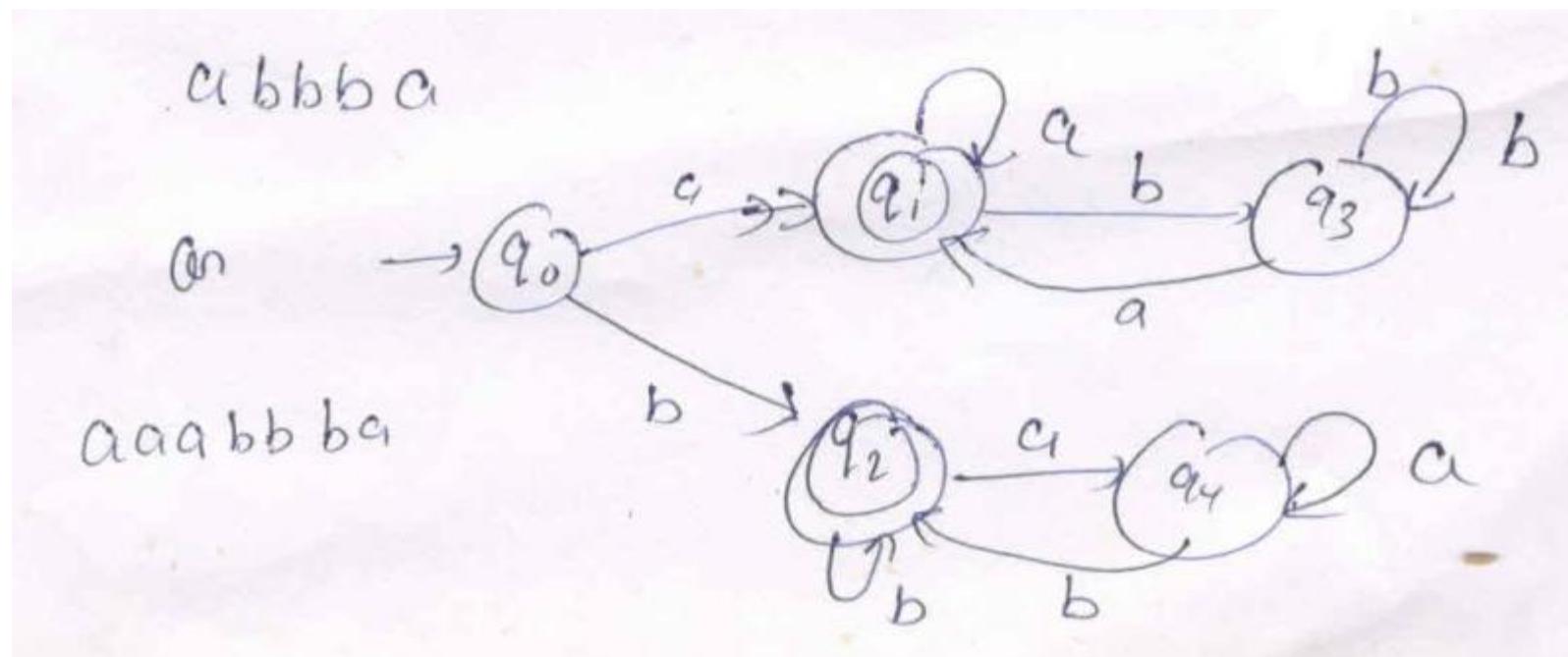
0, 00, 010

01110??

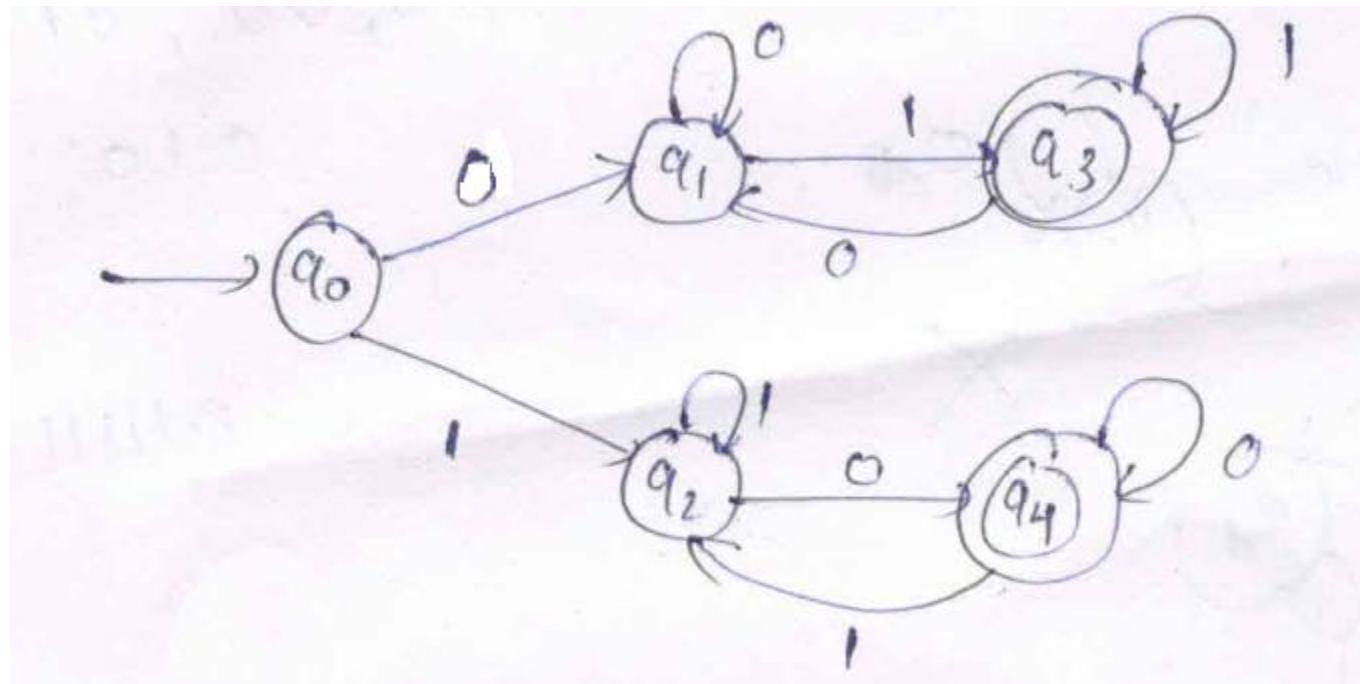
6. Draw a DFA which accepts all the strings on  $\Sigma=\{0,1\}$  which must start & end with '0'.



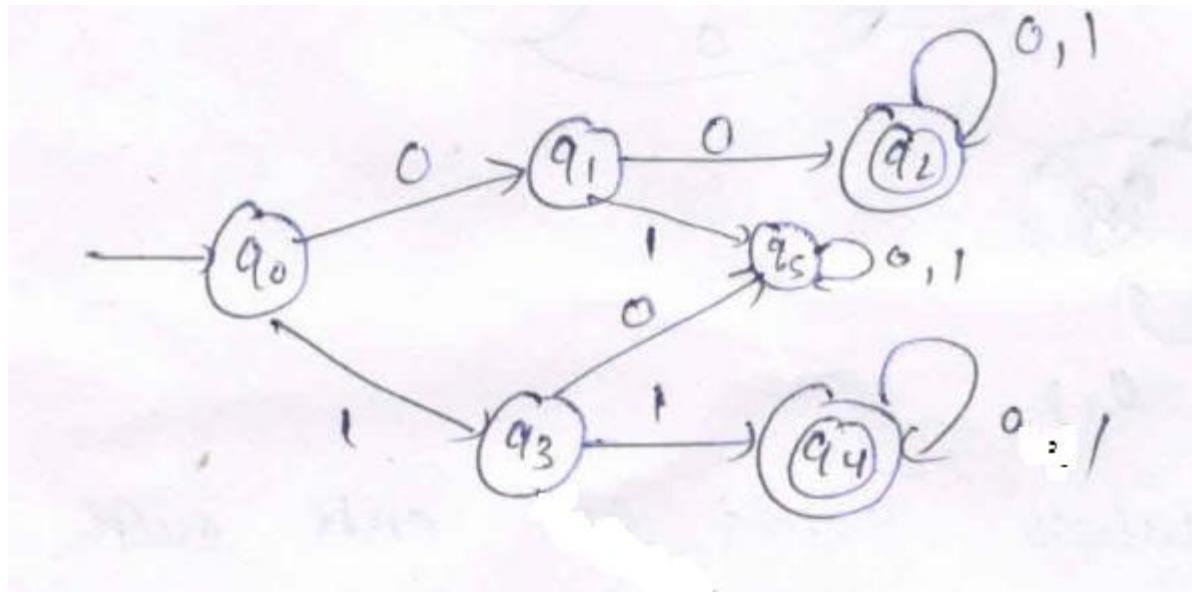
7. Draw a DFA which accepts all the strings on  $\Sigma = \{a, b\}$  which starts and ends with same symbol.



8. Draw a DFA which accepts all the strings on  $\Sigma=\{0,1\}$  which starts and ends with different symbol.



9. Draw a DFA which accepts all the strings on  $\Sigma=\{0,1\}$  which starts with '00' or '11'.



# Practice Problems

1. Draw a DFA which accepts all the strings on  $\Sigma=\{0,1\}$  which never ends with '100'.
2. Draw a DFA which accepts all the strings on  $\Sigma=\{0,1\}$  which contains sub-string '00'.

## Suggested readings

1. An introduction to FORMAL LANGUAGES and AUTOMATA by PETER LINZ.
2. Introduction to Automata Theory, Languages, And Computation by JOHN E. HOPCROFT, RAJEEV MOTWANI, JEFFREY D. ULLMAN
3. Theory of computer science: automata, languages and computation by K.L.P MISHRA

**Thank you**

# Theory of Computation: CS-202

## Deterministic Finite Automata-3

# Content

Finite Automata

    Deterministic Finite Accepters

- Examples

    Non Deterministic Finite Accepters

- Examples

# Examples

1. Draw a DFA over  $\Sigma=\{a,b\}$  which accepts all the strings of length 2.

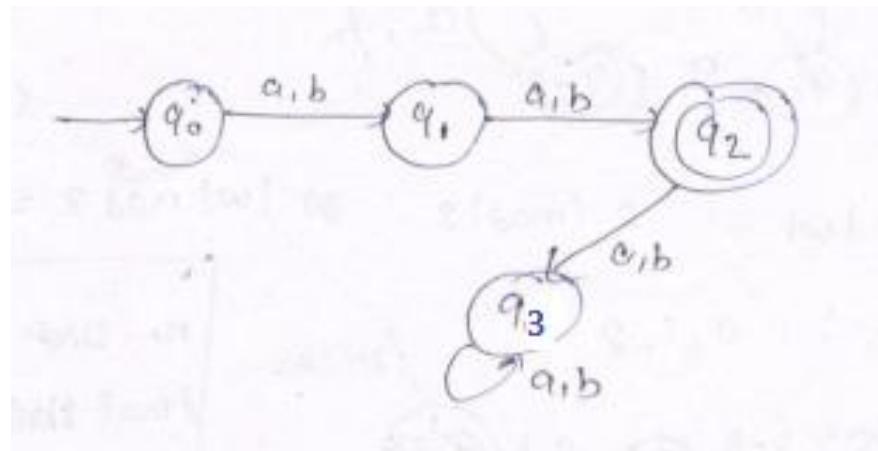
Input strings

b	a	a
---	---	---

not accepted by given DFA.

a	b
---	---

Accepted

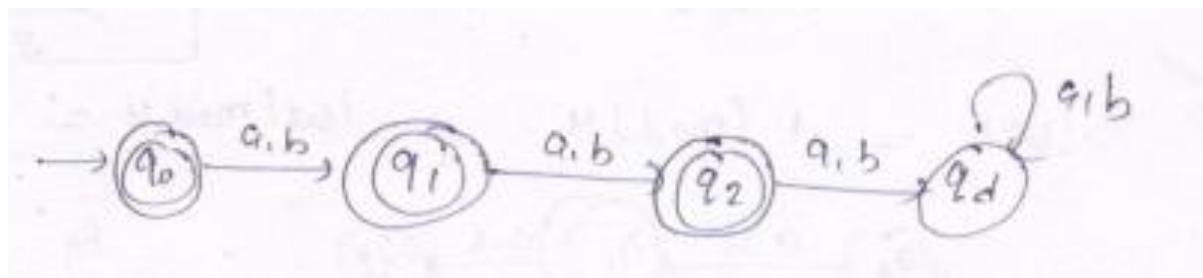


# Example (cont..)

2. Draw a DFA over  $\Sigma = \{a, b\}$  which accepts all the strings of length less or equal to '2'.

Input string      **b**      Accepted by given DFA.

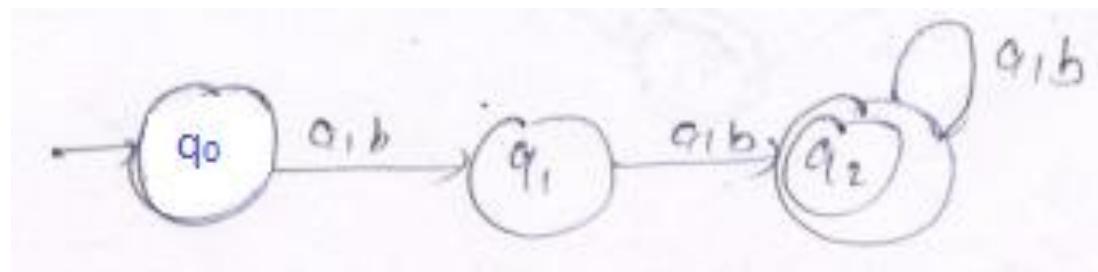
**b** | **a**      Accepted



3. Draw a DFA over  $\Sigma = \{a, b\}$  which accepts all the strings of length greater or equal to '2'.

Input string      

b		a		a		b		b		b
---	--	---	--	---	--	---	--	---	--	---

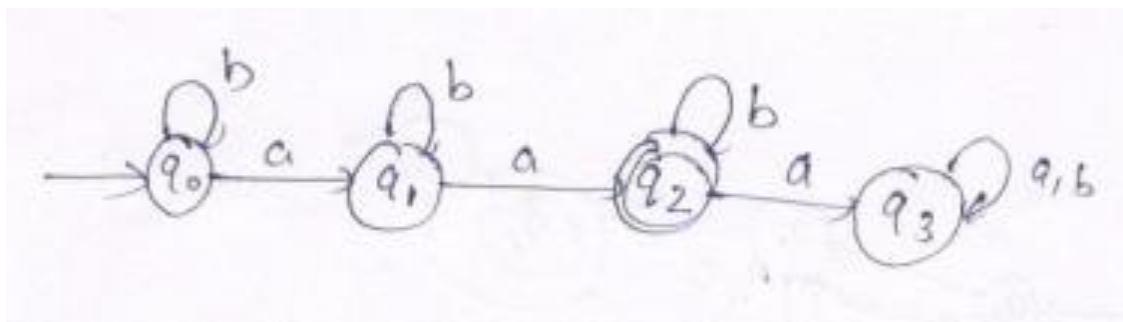


4. Draw a DFA over  $\Sigma = \{a, b\}$  which accepts all the strings 'w' in which  $n_a(w) = 2$ .

Input string    

b	a	a	b	a	b	Not accepted
---	---	---	---	---	---	--------------

b	a	b	b	a	b	Accepted
---	---	---	---	---	---	----------



5. Draw a DFA over  $\Sigma = \{a, b\}$  which accepts all the strings 'w' in which  $n_a(w) \leq 2$ .

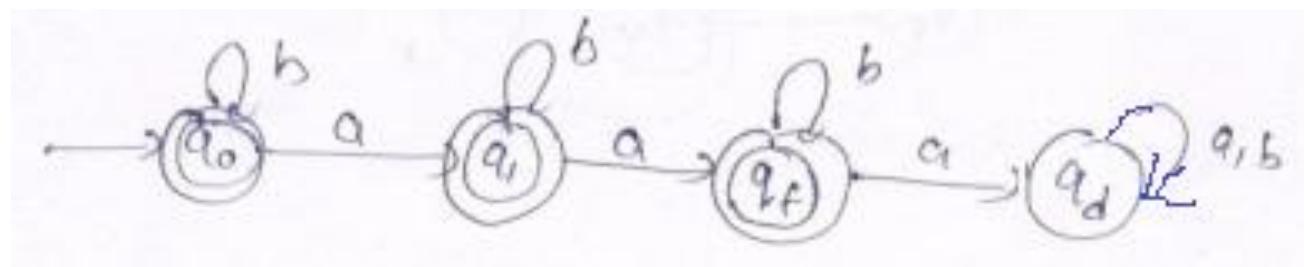
Input string    

b	a	a	b	a	b
---	---	---	---	---	---

    Not accepted

b	a	b	b	b	b
---	---	---	---	---	---

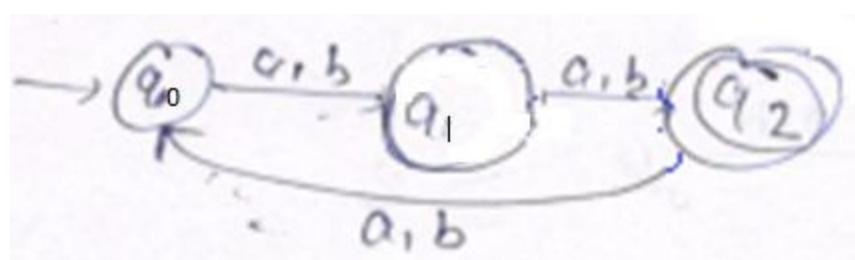
    Accepted



6. Draw a DFA over  $\Sigma = \{a, b\}$  which accepts all the strings 'w' in which  $|w| \bmod 3 = 2$ .

Input string      

b	a	a	b	a
---	---	---	---	---



Mod 3 , Remainder: 0, 1, 2	
String length	Remainder
0	0
1	1
2	2
3	0
4	1
5	2
.	.
.	.

# Nondeterministic Finite Accepters

A nondeterministic finite accepter or nfa is defined by the quintuple

$$M = (Q, \Sigma, \delta, q_0, F)$$

where  $Q$  is a finite set of internal states,

$\Sigma$  is a finite set of symbols called the input alphabet,

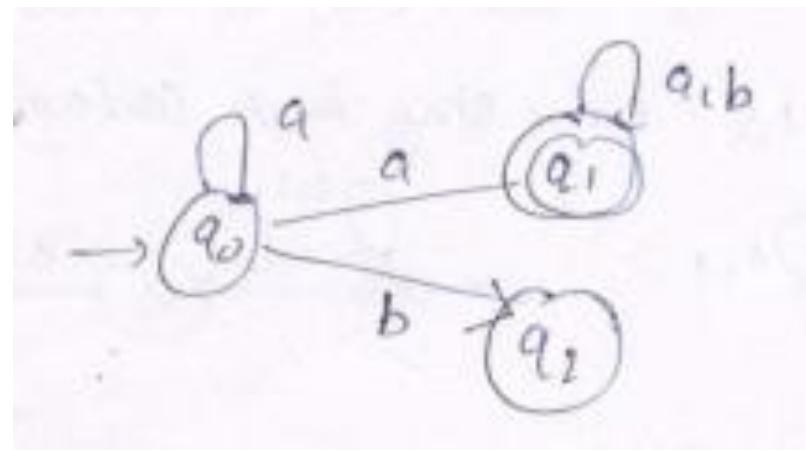
$q_0 \in Q$  is the initial state,

$F \subseteq Q$  is a set of final states.

$$\delta: Q \times \Sigma \rightarrow 2^Q$$

$\forall$  NFA  $\exists$  a DFA

$\Rightarrow$  DFA  $\subseteq$  NFA



$$\delta(q_0, a) = \{q_0, q_1\}$$

$$\delta(q_2, b) = \{ \lambda \}$$

# Transition graph

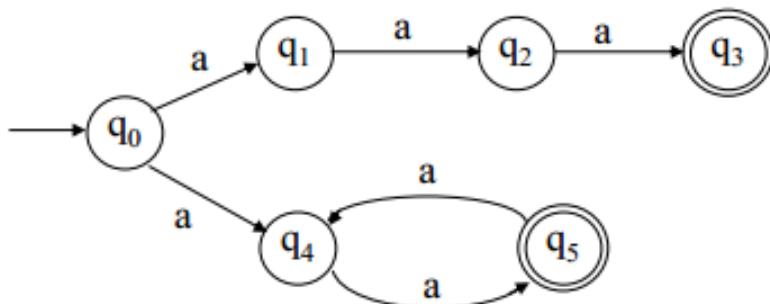
Transition Graph of an nfa  $M = (Q, \Sigma, \delta, q_0, F)$

Vertex labeled with  $q_i$ : state  $q_i \in Q$ ,

Edge from  $q_i$  to  $q_j$  labeled with  $a$ :  $q_j \in \delta(q_i, a)$

Example

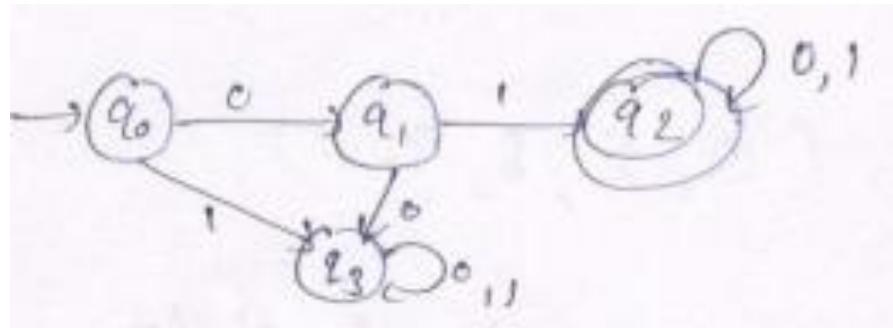
An nfa is shown as below



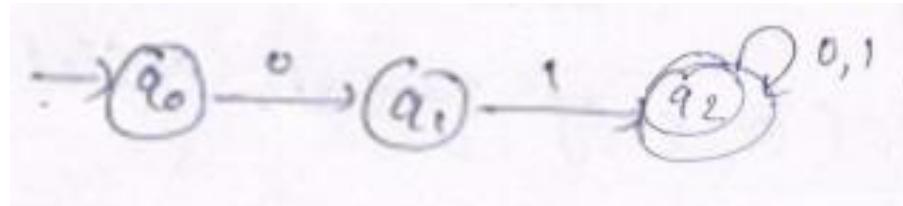
# Examples

1. Draw NFA which accepts all the strings on  $\Sigma=\{0,1\}$  which starts with '01'.

DFA



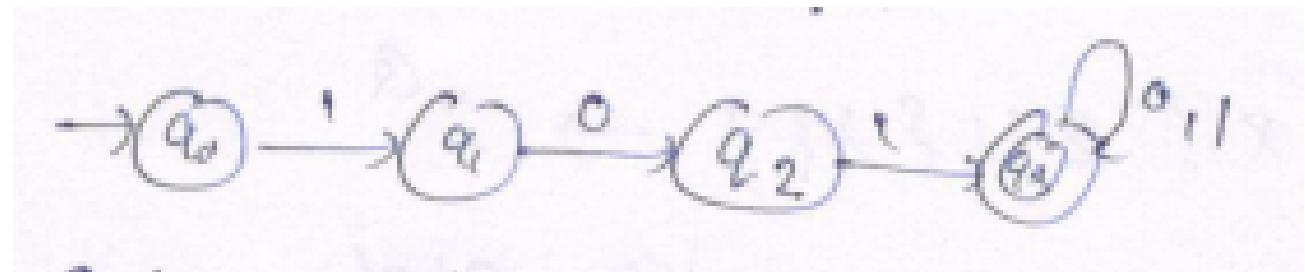
NFA



# Examples

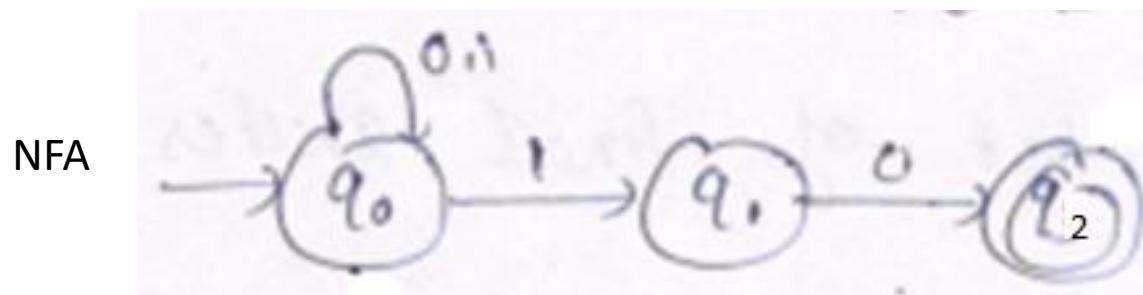
2. Design NFA which accepts all the strings on  $\Sigma=\{0,1\}$  starting with '101'.

NFA



# Examples

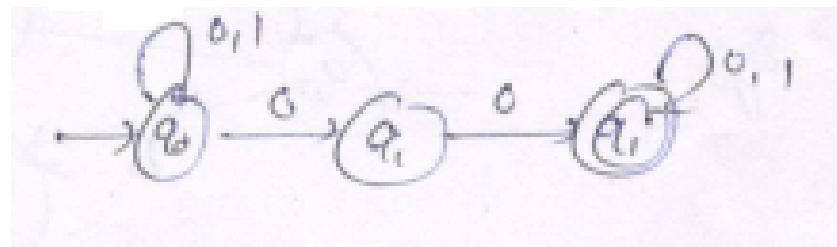
3. Design NFA which accepts all the strings on  $\Sigma=\{0,1\}$  ending with '10'.



# Examples

4. Design NFA which accepts all the strings on  $\Sigma=\{0,1\}$  which contains substring ‘00’.

NFA



## Suggested readings

1. An introduction to FORMAL LANGUAGES and AUTOMATA by PETER LINZ.
2. Introduction to Automata Theory, Languages, And Computation by JOHN E. HOPCROFT, RAJEEV MOTWANI, JEFFREY D. ULLMAN
3. Theory of computer science: automata, languages and computation by K.L.P MISHRA

**Thank you**

# Theory of Computation: CS-202

## Non Deterministic Finite Automata

# Content

## Finite Automata

- Deterministic Finite Accepters
- Definition of a Nondeterministic Acceptor
- Why Nondeterministic
- Equivalence of Deterministic and Nondeterministic Finite Accepters

# Nondeterministic Finite Accepters

A nondeterministic finite accepter or nfa is defined by the quintuple

$$M = (Q, \Sigma, \delta, q_0, F)$$

where  $Q$  is a finite set of internal states,

$\Sigma$  is a finite set of symbols called the input alphabet,

$q_0 \in Q$  is the initial state,

$F \subseteq Q$  is a set of final states.

$$\delta: Q \times \Sigma \rightarrow 2^Q$$

$\forall$  NFA  $\exists$  a DFA

$\Rightarrow$  DFA  $\subseteq$  NFA

# Conversion from NFA to DFA

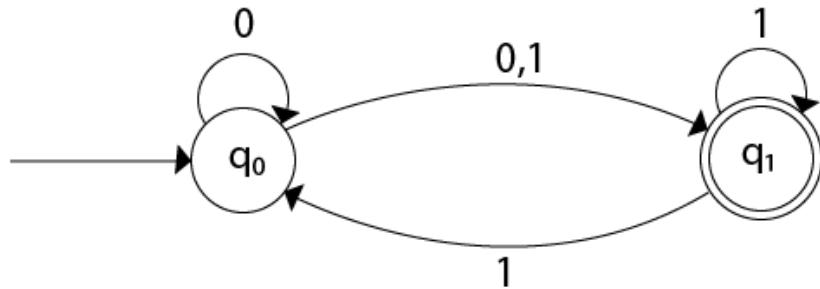
- In NFA, when a specific input is given to the current state, the machine goes to multiple states. It can have zero, one or more than one move on a given input symbol.
- On the other hand, in DFA, when a specific input is given to the current state, the machine goes to only one state. DFA has only one move on a given input symbol.
- Let,  $M = (Q, \Sigma, \delta, q_0, F)$  is an NFA which accepts the language  $L(M)$ . There should be equivalent DFA denoted by  $M' = (Q', \Sigma', q_0', \delta', F')$  such that  $L(M) = L(M')$ .

# Steps for converting NFA to DFA:

- **Step 1:** Initially  $Q' = \emptyset$
- **Step 2:** Add  $q_0$  of NFA to  $Q'$ . Then find the transitions from this start state.
- **Step 3:** In  $Q'$ , find the possible set of states for each input symbol. If this set of states is not in  $Q'$ , then add it to  $Q'$ .
- **Step 4:** In DFA, the final state will be all the states which contain  $F$ (final states of NFA)

# Example-1

Convert the given NFA to DFA



Solution: For the given transition diagram we will first construct the transition table.

State	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_1\}$
$*q_1$	$\emptyset$	$\{q_0, q_1\}$

# Example-1 (Cont..)

Solution:

State	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_1\}$
$*q_1$	$\emptyset$	$\{q_0, q_1\}$

Now we will obtain  $\delta'$  transition for state  $q_0$ .

$$\begin{aligned}\delta'([q_0], 0) &= \{q_0, q_1\} \\ &= [q_0, q_1] \quad (\text{new state})\end{aligned}$$

$$\delta'([q_0], 1) = \{q_1\} = [q_1]$$

# Example-1 (Cont..)

Solution:

State	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_1\}$
$*q_1$	$\emptyset$	$\{q_0, q_1\}$

Now we will obtain  $\delta'$  transition for state  $q_1$ .

$$\delta'([q_1], 0) = \emptyset, \delta'([q_1], 0) = q_2 \text{ (new state)}$$

$$\delta'([q_1], 1) = [q_0, q_1]$$

# Example-1 (Cont..)

Solution:

State	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_1\}$
$*q_1$	$\phi$	$\{q_0, q_1\}$

Note: for  $\delta'([q_1], 0) = \phi$ , we need to create a new state (called dead state) say  $q_2$   
Now we will obtain  $\delta'$  transition for state  $q_2$ .

$$\delta'([q_2], 0) = q_2$$

$$\delta'([q_2], 1) = q_2$$

# Example-1 (Cont..)

Solution:

State	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_1\}$
$*q_1$	$\emptyset$	$\{q_0, q_1\}$

Now we will obtain  $\delta'$  transition on  $[q_0, q_1]$ .

$$\delta'([q_0, q_1], 0) = \delta(q_0, 0) \cup \delta(q_1, 0)$$

$$= \{q_0, q_1\} \cup \emptyset$$

$$= \{q_0, q_1\}$$

$$= [q_0, q_1]$$

$$\delta'([q_0, q_1], 1) = \delta(q_0, 1) \cup \delta(q_1, 1)$$

$$= \{q_1\} \cup \{q_0, q_1\}$$

$$= \{q_0, q_1\}$$

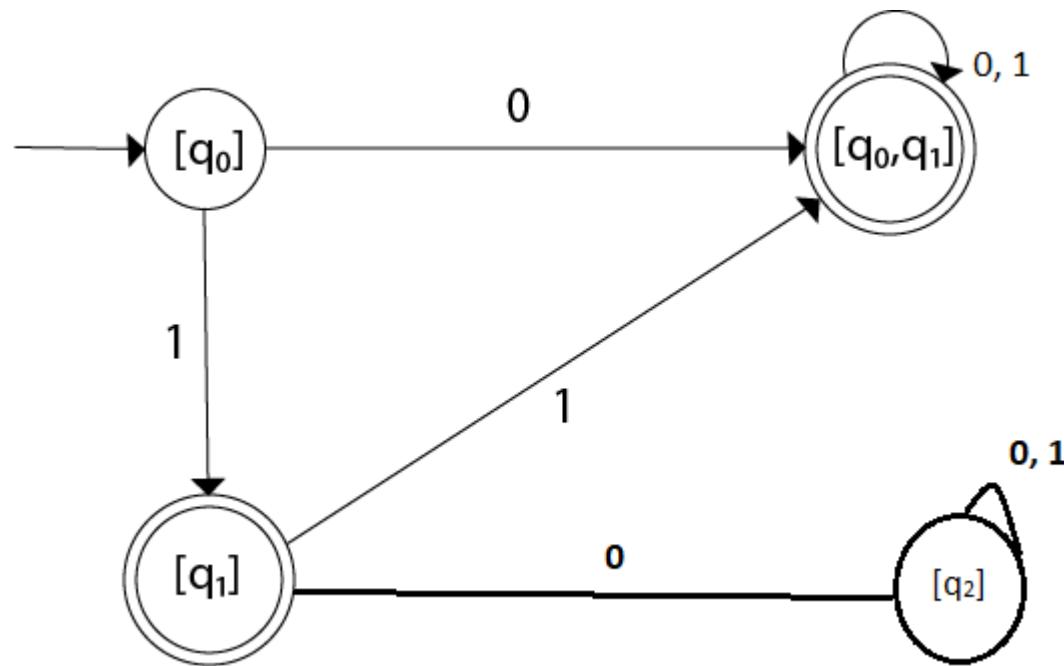
$$= [q_0, q_1]$$

- As in the given NFA,  $q_1$  is a final state, then in DFA wherever,  $q_1$  exists that state becomes a final state. Hence in the DFA, final states are  $[q_1]$  and  $[q_0, q_1]$ .
- Therefore set of final states  $F = \{[q_1], [q_0, q_1]\}$ .

- The transition table for the constructed DFA will be:

State	0	1
$\rightarrow[q_0]$	$[q_0, q_1]$	$[q_1]$
$*[q_1]$	$[q_2]$	$[q_0, q_1]$
$*[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$
$q_2$	$q_2$	$[q_2]$

The Transition diagram for DFA will be:



# $\epsilon$ –NFA or Epsilon NFA

It is an extended version of NFA, which makes the designing of NFA much easier.

An  $\epsilon$  –NFA is defined using the quintuple

$$M = (Q, \Sigma, q_0, \delta, F)$$

where  $Q$  is a finite set of **internal states**,

$\Sigma$  is a finite set of symbols called the **input alphabet**,

$q_0 \in Q$  is the **initial state**,

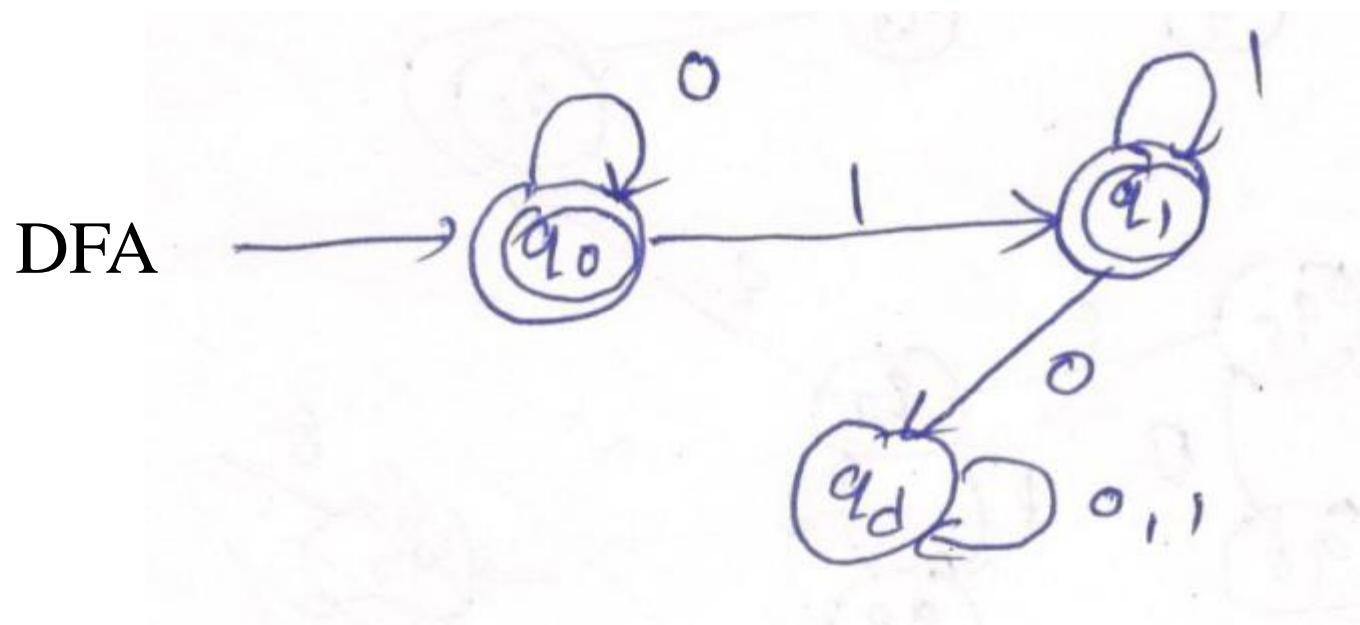
$F \subseteq Q$  is a set of **final states**.

$$\delta: Q \times (\Sigma \cup \epsilon) \rightarrow 2^Q$$

# Examples

1. Design an  $\epsilon$  NFA for the language

$$L = \{0^n 1^m, n, m \geq 0\}$$

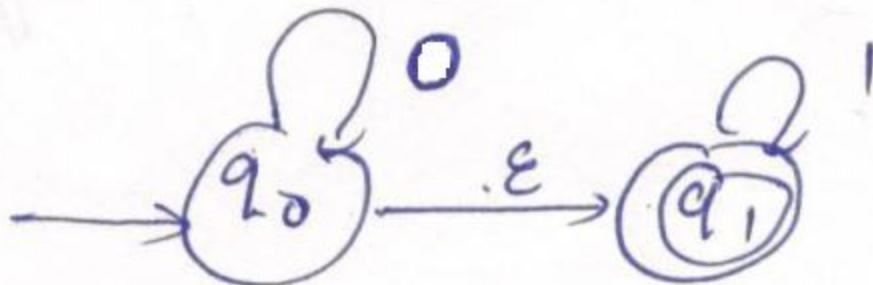


# Examples (Cont..)

1. Design an  $\epsilon$  NFA for the language

$$L = \{0^n 1^m, n, m \geq 0\}$$

$\epsilon$  NFA



## Suggested readings

1. An introduction to FORMAL LANGUAGES and AUTOMATA by PETER LINZ.
2. Introduction to Automata Theory, Languages, And Computation by JOHN E. HOPCROFT, RAJEEV MOTWANI, JEFFREY D. ULLMAN
3. Theory of computer science: automata, languages and computation by K.L.P MISHRA

**Thank you**

# Theory of Computation: CS-202

## Non Deterministic Finite Automata

# Outlines

## Nondeterministic Finite Accepters

- Epsilon Nondeterministic Finite Accepters ( $\epsilon$  –NFA )
- Conversion from  $\epsilon$  –NFA to NFA

# $\epsilon$ –NFA or Epsilon NFA

It is an extended version of NFA, which makes the designing of NFA much easier.

An  $\epsilon$  –NFA is defined using the quintuple

$$M = (Q, \Sigma, q_0, \delta, F)$$

where  $Q$  is a finite set of **internal states**,

$\Sigma$  is a finite set of symbols called the **input alphabet**,

$q_0 \in Q$  is the **initial state**,

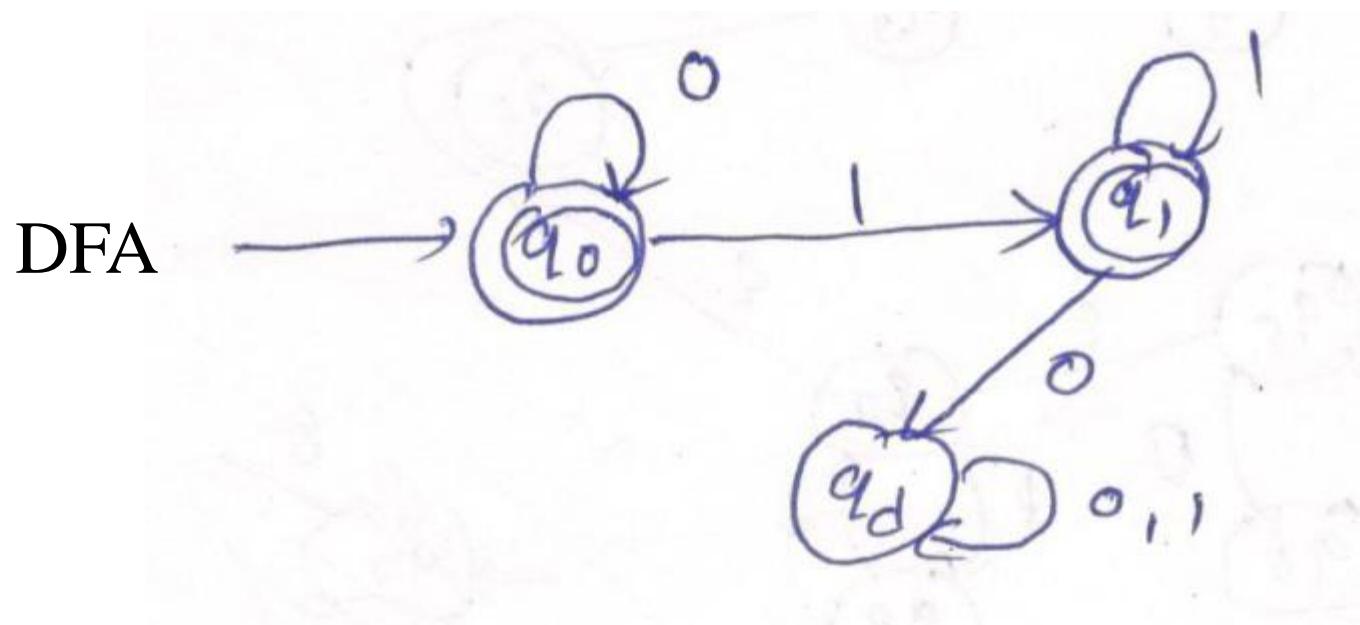
$F \subseteq Q$  is a set of **final states**.

$$\delta: Q \times (\Sigma \cup \epsilon) \rightarrow 2^Q$$

# Examples

1. Design an  $\epsilon$  NFA for the language

$$L = \{0^n 1^m, n, m \geq 0\}$$

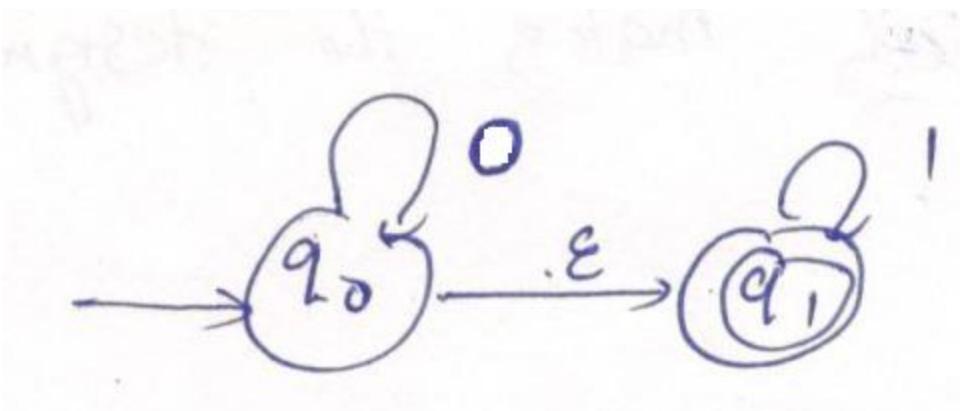


# Examples (Cont..)

1. Design an  $\epsilon$  NFA for the language

$$L = \{0^n 1^m, n, m \geq 0\}$$

$\epsilon$  NFA



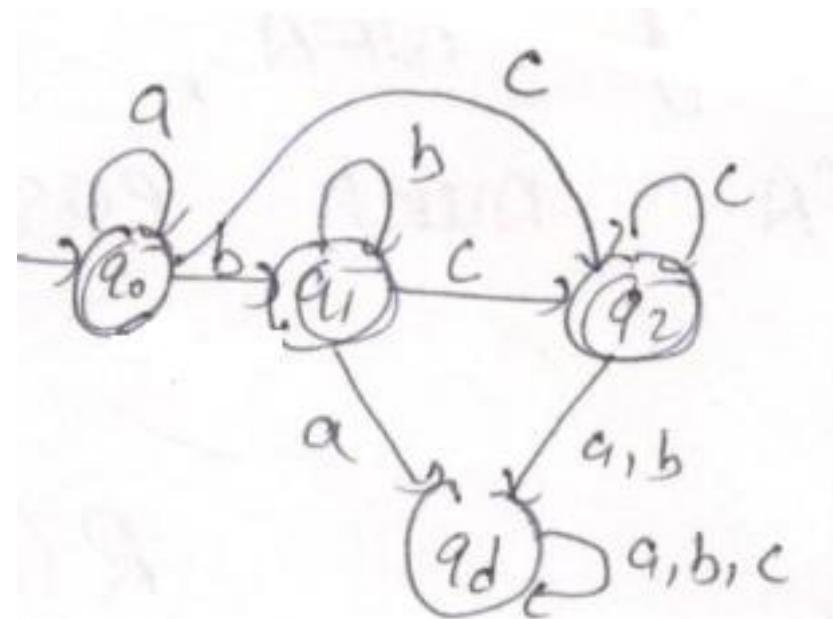
# Examples (Cont..)

2. Design an  $\epsilon$  NFA for the language

$$L = \{a^m b^n c^p, m, n, p \geq 0\}$$

So first design DFA

DFA

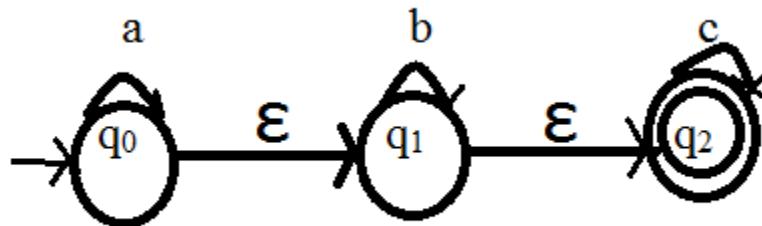


# Examples (Cont..)

2. Design an  $\epsilon$  NFA over  $\Sigma = \{a, b, c\}$  for the language

$$L = \{a^m b^n c^p, m, n, p \geq 0\}$$

$\epsilon$  NFA



# Examples (Cont..)

3. Design an  $\epsilon$  NFA for the language

$$L = \{0^m 1^n, m+n=\text{odd}\}$$

Occurrence of 0's and 1's

0	1	sum
Even	even	even
Even	odd	odd
Odd	even	odd
Odd	odd	even

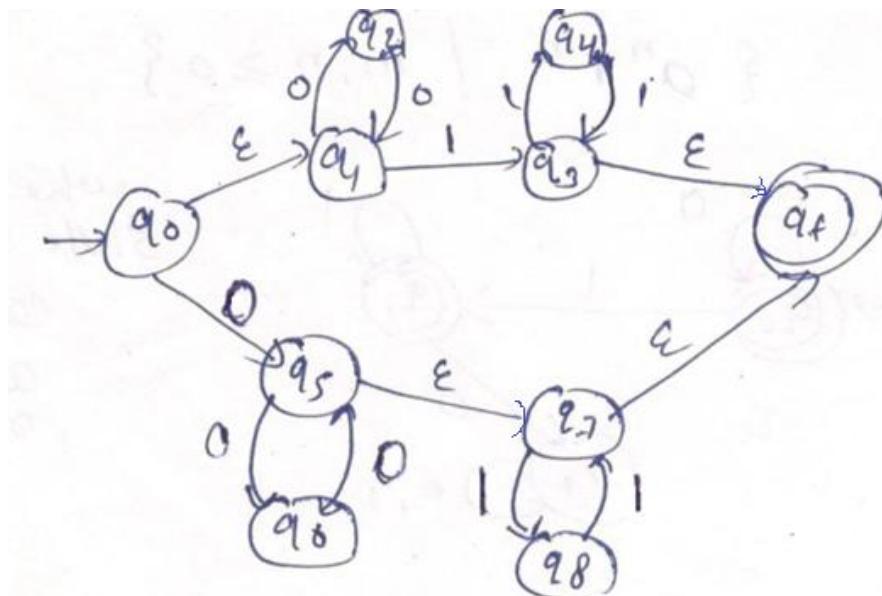
# Examples (Cont..)

## 3. Design an $\epsilon$ NFA for the language

$$L = \{0^m 1^n, m+n=\text{odd}\}$$

Input Strings

0   0   1   1	Rejected
0   0   1   1   1	Accepted



# Conversion from $\epsilon$ -NFA to NFA

- In NFA, when a specific input is given to the current state, the machine goes to multiple states. It can have zero, one or more than one move on a given input symbol.
- Epsilon NFA is the NFA which contains epsilon move(s)/Null move(s). To remove the epsilon move/Null move from epsilon-NFA we can convert it into NFA.

## **$\varepsilon$ -closure:**

It is a set of all the states which can be obtained from a state, only by seeing  $\varepsilon$ -moves.

Now transition function for NFA is defined as:

$$\delta'(q_i, x) = \varepsilon\text{-closure}[\delta(\varepsilon\text{-closure}(q_i), x)]$$

x is an input alphabet

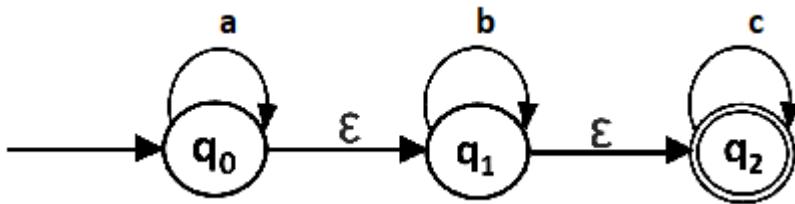
# Steps for converting $\epsilon$ -NFA to NFA:

- **Step-1:** Consider the vertices having the epsilon move. For each state (or vertex) find epsilon closure.
- **Step-2:** Now obtain  $\delta'$  transition function of NFA for the states of  $\epsilon$ -NFA over each input symbol ‘x’ (say).  
$$\delta'(q_i, x) = \epsilon\text{-closure}[\delta(\epsilon\text{-closure}(q_i), x)]$$

x is an input alphabet
- **Step-3:** In NFA, the final state will be all the states whose closure contain F(final states of  $\epsilon$ -NFA)

# Example

Convert the given  $\epsilon$  NFA to NFA



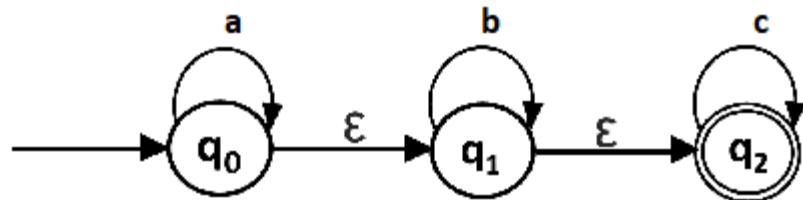
Solution: For the given transition diagram we will first obtain the  $\epsilon$ -closure of each state.

$$\epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\}$$

$$\epsilon\text{-closure}(q_1) = \{q_1, q_2\}$$

$$\epsilon\text{-closure}(q_2) = \{q_2\}$$

# Example (Cont..)



$$\delta'(q_i, x) = \text{ε-closure}[\delta(\text{ε-closure}(q_i), x)]$$

Solution:

Now we will obtain  $\delta'$  transition. Let  $\text{ε-closure}(q_0) = \{q_0, q_1, q_2\}$

$$\begin{aligned}
 \delta'(q_0, a) &= \text{ε-closure}\{\delta((q_0, q_1, q_2), a)\} &= \text{ε-closure}\{\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a)\} \\
 &= \text{ε-closure}\{q_0\} &= \{q_0, q_1, q_2\}
 \end{aligned}$$

$$\begin{aligned}
 \delta'(q_0, b) &= \text{ε-closure}\{\delta((q_0, q_1, q_2), b)\} &= \text{ε-closure}\{\delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b)\} \\
 &= \text{ε-closure}\{q_1\} &= \{q_1, q_2\}
 \end{aligned}$$

$$\begin{aligned}
 \delta'(q_0, c) &= \text{ε-closure}\{\delta((q_0, q_1, q_2), c)\} &= \text{ε-closure}\{\delta(q_0, c) \cup \delta(q_1, c) \cup \delta(q_2, c)\} \\
 &= \text{ε-closure}\{q_2\} &= \{q_2\}
 \end{aligned}$$

# Example (Cont..)

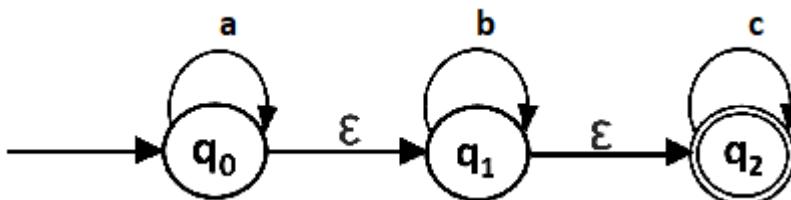
Solution:

Thus we have obtained

$$\delta'(q_0, a) = \{q_0, q_1, q_2\}$$

$$\delta'(q_0, b) = \{q_1, q_2\}$$

$$\delta'(q_0, c) = \{q_2\}$$



$$\delta'(q_i, x) = \text{ε-closure}[\delta(\text{ε-closure}(q_i), x)]$$

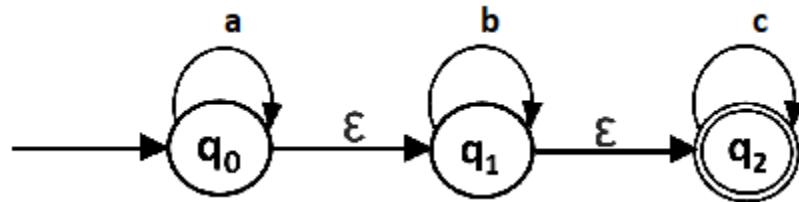
- Now we will find the transitions on states  $q_1$  and  $q_2$  for each input.

Hence

$$\begin{aligned}\delta'(q_1, a) &= \text{ε-closure}\{\delta(q_1, a)\} &= \text{ε-closure}\{\delta(q_1, a) \cup \delta(q_2, a)\} \\ &= \text{ε-closure}\{\varphi\} &= \varphi\end{aligned}$$

# Example (Cont..)

Solution:



$$\delta'(q_i, x) = \text{ε-closure}[\delta(\text{ε-closure}(q_i), x)]$$

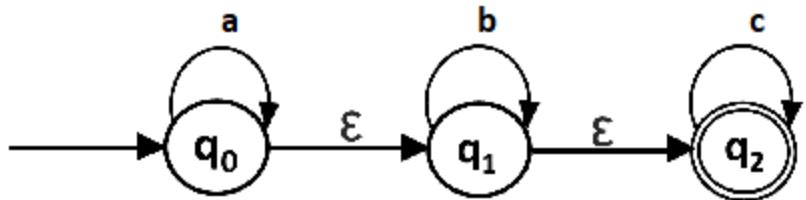
$$\begin{aligned}\delta'(q_1, b) &= \text{ε-closure}\{\delta(q_1, q_2), b\} & &= \text{ε-closure}\{\delta(q_1, b) \cup \delta(q_2, b)\} \\ & & &= \text{ε-closure}\{q_1\} & &= \{q_1, q_2\}\end{aligned}$$

$$\begin{aligned}\delta'(q_1, c) &= \text{ε-closure}\{\delta(q_1, q_2), c\} & &= \text{ε-closure}\{\delta(q_1, c) \cup \delta(q_2, c)\} \\ & & &= \{q_2\}\end{aligned}$$

- Thus we have obtained
- $\delta'(q_1, a) = \varphi$
- $\delta'(q_1, b) = \{q_1, q_2\}$
- $\delta'(q_1, c) = \{q_2\}$

# Example (Cont..)

Solution:



$$\delta'(q_i, x) = \text{ε-closure}[\delta(\text{ε-closure}(q_i), x)]$$

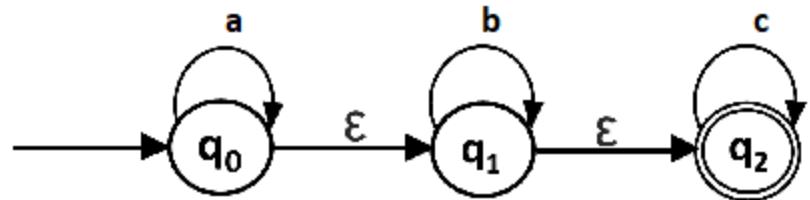
- Now we will obtain transitions for  $q_2$ :

$$\begin{aligned}\delta'(q_2, a) &= \text{ε-closure}\{\delta(q_2, a)\} &= \text{ε-closure}\{\varphi\} &= \varphi \\ \delta'(q_2, b) &= \text{ε-closure}\{\delta(q_2, b)\} &= \text{ε-closure}\{\varphi\} &= \varphi \\ \delta'(q_2, c) &= \text{ε-closure}\{\delta(q_2, c)\} &= \{q_2\}\end{aligned}$$

As  $\text{ε-closure}\{q_0\} = \{q_0, q_1, q_2\}$  in which final state  $q_2$  lies hence  $q_0$  is final state.

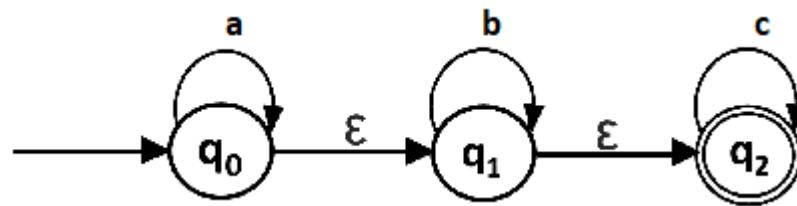
$\text{ε-closure}\{q_1\} = \{q_1, q_2\}$  in which the state  $q_2$  lies hence  $q_2$  is also final state.

$\text{ε-closure}\{q_2\} = \{q_2\}$ , the state  $q_2$  lies hence  $q_2$  is also a final state.

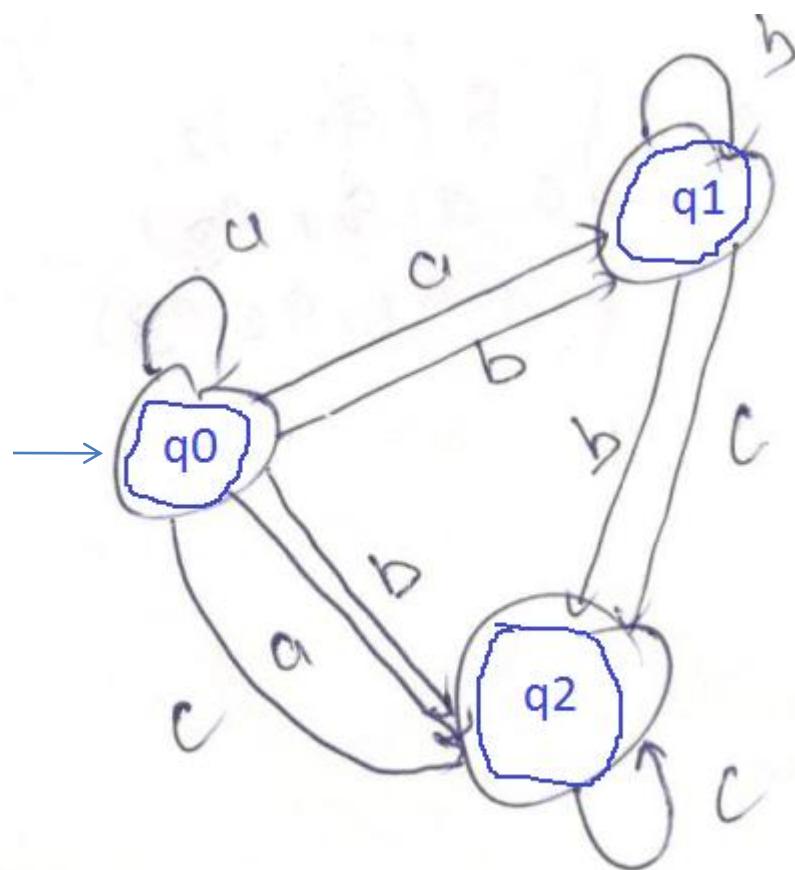


The Transition table for NFA will be:

States	a	b	c
$\xrightarrow{*} q_0$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
$*q_1$	$\varphi$	$\{q_1, q_2\}$	$\{q_2\}$
$*q_2$	$\varphi$	$\varphi$	$\{q_2\}$



The Transition diagram for NFA will be:



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

## Suggested readings

1. An introduction to FORMAL LANGUAGES and AUTOMATA by PETER LINZ.
2. Introduction to Automata Theory, Languages, And Computation by JOHN E. HOPCROFT, RAJEEV MOTWANI, JEFFREY D. ULLMAN
3. Theory of computer science: automata, languages and computation by K.L.P MISHRA

# Thank you

# Theory of Computation: CS-202

## Deterministic Finite Automata Minimization

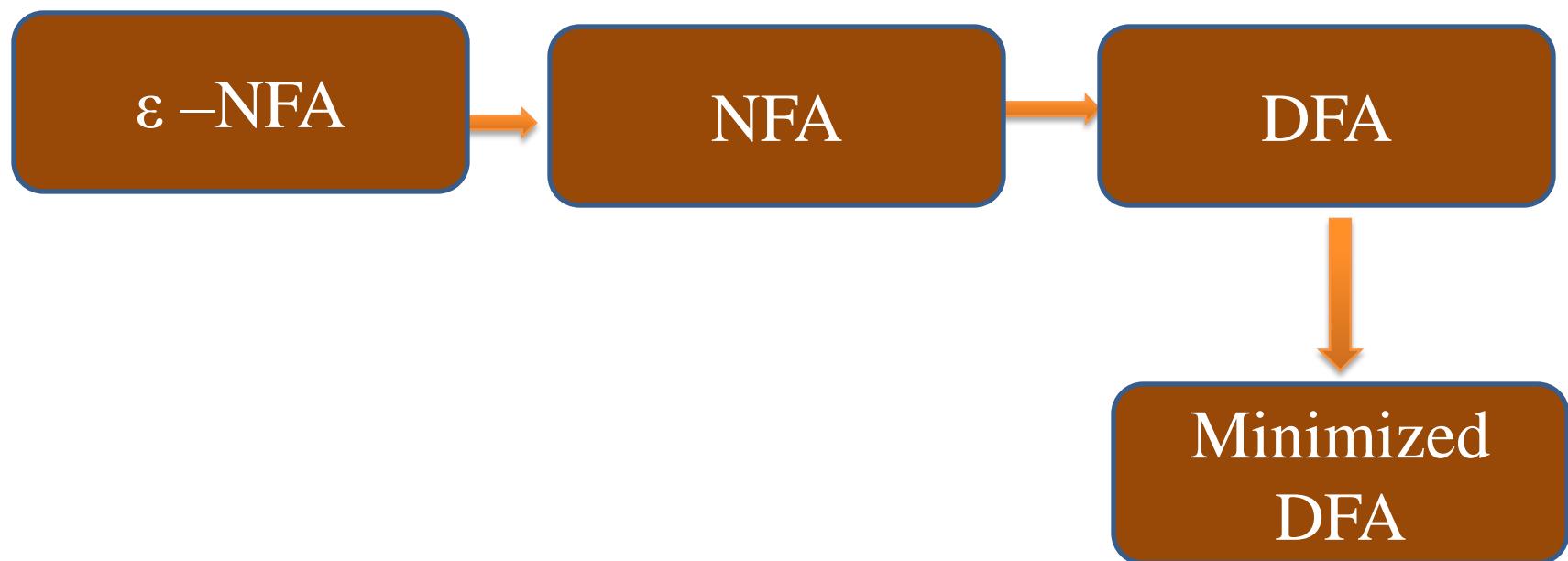
# Outlines

Deterministic Finite Accepters (DFA)

□ Minimization of DFA

➤ Equivalence Theorem or Set Partitioning method

# Finite Automata



# Minimization of DFA

- Minimization of DFA means reducing the number of states from the given DFA.

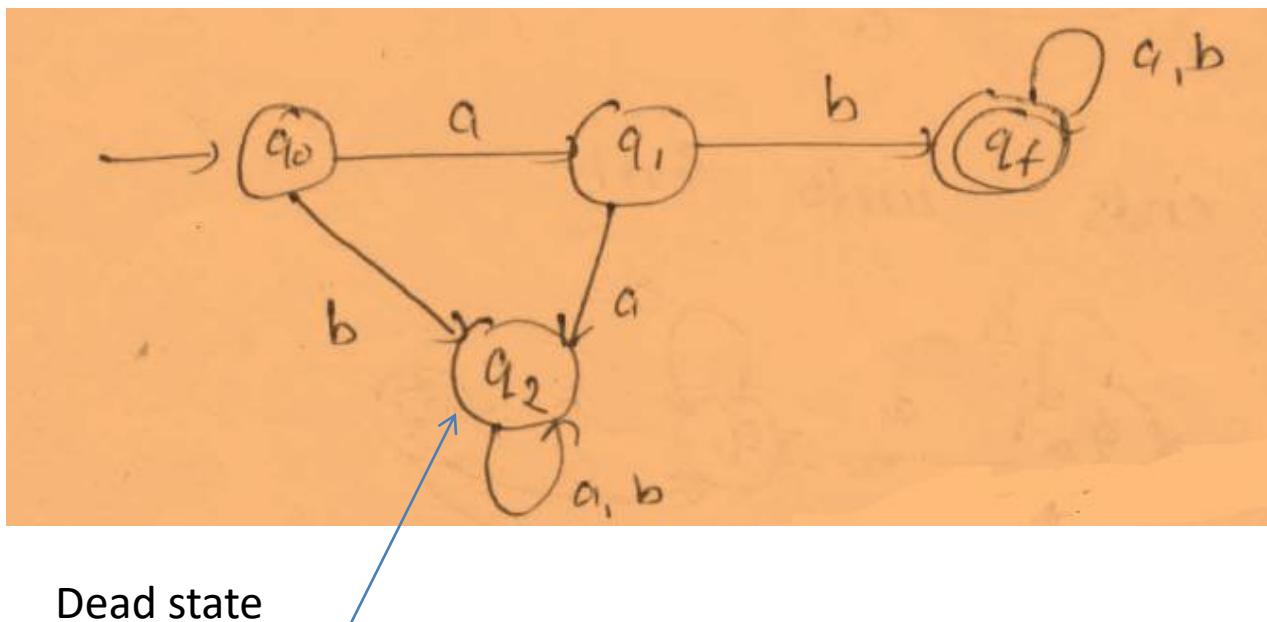
# Minimization of DFA

DFA minimization is possible by considering following states:

1. Dead states
2. Unreachable states
3. Equal or indistinguishable states

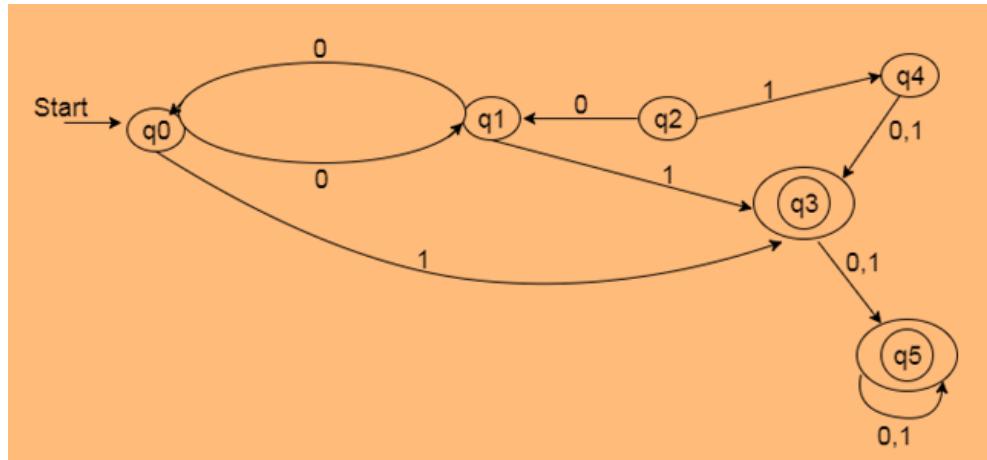
# Minimization of DFA

- **Dead states:** A dead state is non accepting state whose transitions for every input symbols terminate on themselves.
- There is no way for dead state to reach a final state.
- Merge all the dead states of DFA.



# Minimization of DFA

- **Unreachable states:** Unreachable states are the states that are not reachable from the initial state of the DFA for any input string.
- Remove unreachable states from DFA.



q2 and q4 are unreachable

# Minimization of DFA

- **Equal or indistinguishable states:** Two states  $(q_i, q_j)$  of a DFA are said to be equal or indistinguishable if
$$\delta^*(q_i, w) \in F \Rightarrow \delta^*(q_j, w) \in F$$

And

$$\delta^*(q_i, w) \notin F \Rightarrow \delta^*(q_j, w) \notin F$$

$\forall w \in \Sigma^*$ ,  $q_i, q_j$  are called equal or indistinguishable.

$\exists w \in \Sigma^*$  such that

$$\delta^*(q_i, w) \in F \Rightarrow \delta^*(q_j, w) \notin F$$

Then  $q_i, q_j$  are called unequal or distinguishable.

Note: merge the equal states.

# DFA Minimization using Equivalence Theorem or Set Partitioning method

If  $q_i$  and  $q_j$  are two states in a DFA, we can combine these two states into  $\{q_i, q_j\}$  if they are not distinguishable or equal.

Two states are distinguishable, if there is at least one string ‘w’, such that one of  $\delta(q_i, w)$  and  $\delta(q_j, w)$  is accepting and another is not accepting.

Hence, a DFA is minimal if and only if all the states are distinguishable.

## Steps for DFA Minimization using Equivalence Theorem or Set Partitioning method

**Step 1** – All the states  $Q$  are divided in two partitions – **final states** and **non-final states** and are denoted by  $P_0$ . All the states in a partition are  $0^{\text{th}}$  equivalent. Take a counter  $k$  and initialize it with 0.

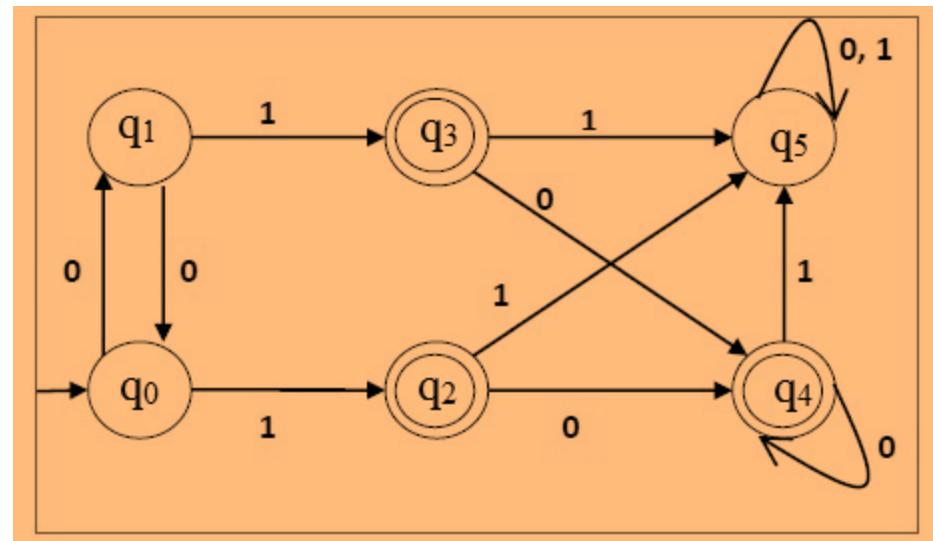
**Step 2** – Increment  $k$  by 1. For each partition in  $P_k$ , divide the states in  $P_k$  into two partitions if they are  $k$ -distinguishable. Two states within this partition  $q_i$  and  $q_j$  are  $k$ -distinguishable if there is an input  $w$  such that  $\delta(q_i, w)$  and  $\delta(q_j, w)$  are  $(k-1)$  distinguishable.

**Step 3** – If  $P_k \neq P_{k-1}$ , repeat Step 2, otherwise go to Step 4.

**Step 4** – Combine  $k^{\text{th}}$  equivalent sets and make them the new states of the reduced DFA.

Note: merge dead states and remove unreachable states before following these steps. 10

Example: Minimize the DFA using set partitioning method.



# Solution

States ( $q_i$ )	$\delta(q_i, 0)$	$\delta(q_i, 1)$
$\rightarrow q_0$	q1	q2
$q_1$	q0	q3
$*q_2$	q4	q5
$*q_3$	q4	q5
$*q_4$	q4	q5
$q_5$	q5	q5

# Solution (cont..)

States ( $q$ )	$\delta(q_i, 0)$	$\delta(q_i, 1)$
$\xrightarrow{} q_0$	$q_1$	$q_2$
$q_1$	$q_0$	$q_3$
$*q_2$	$q_4$	$q_5$
$*q_3$	$q_4$	$q_5$
$*q_4$	$q_4$	$q_5$
$q_5$	$q_5$	$q_5$

Now, apply the equivalence theorem to the DFA:

$$P_0 = \{(q2, q3, q4), (q0, q1, q5)\}$$

$$P_1 = \{(q2, q3, q4), (q0, q1), (q5)\}$$

$$P_2 = \{(q2, q3, q4), (q0, q1), (q5)\}$$

Hence,  $P_1 = P_2$ .

**Check 1-equivalence of ( $q_2, q_3$ )**

$$\delta(q_2, 0) = q_4 \text{ and } \delta(q_3, 0) = q_4$$

$$\delta(q_2, 1) = q_5 \text{ and } \delta(q_3, 1) = q_5$$

**Check 1-equivalence of ( $q_2, q_4$ )**

$$\delta(q_2, 0) = q_4 \text{ and } \delta(q_4, 0) = q_4$$

$$\delta(q_2, 1) = q_5 \text{ and } \delta(q_4, 1) = q_5$$

**Check 1-equivalence of ( $q_3, q_4$ )**

$$\delta(q_3, 0) = q_4 \text{ and } \delta(q_4, 0) = q_4$$

$$\delta(q_3, 1) = q_5 \text{ and } \delta(q_4, 1) = q_5$$

# Solution (cont..)

States ( $q$ )	$\delta(q_i, 0)$	$\delta(q_i, 1)$
$\xrightarrow{} q_0$	$q_1$	$q_2$
$q_1$	$q_0$	$q_3$
$*q_2$	$q_4$	$q_5$
$*q_3$	$q_4$	$q_5$
$*q_4$	$q_4$	$q_5$
$q_5$	$q_5$	$q_5$

Now, apply the equivalence theorem to the DFA:

$$P_0 = \{(q2, q3, q4), (q0, q1, q5)\}$$

$$P_1 = \{(q2, q3, q4), (q0, q1), (q5)\}$$

$$P_2 = \{(q2, q3, q4), (q0, q1), (q5)\}$$

Hence,  $P_1 = P_2$ .

**Check 1-equivalence of  $(q0, q1)$**

$$\delta(q0, 0) = q1 \text{ and } \delta(q1, 0) = q0$$

$$\delta(q0, 1) = q2 \text{ and } \delta(q1, 1) = q3$$

**Check 1-equivalence of  $(q0, q5)$**

$$\delta(q0, 0) = q1 \text{ and } \delta(q5, 0) = q5$$

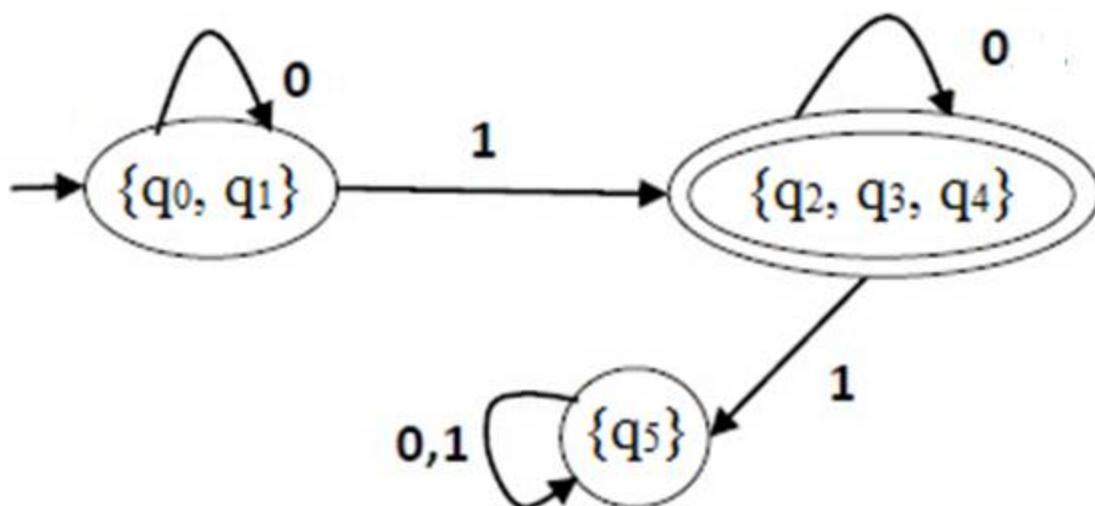
$$\delta(q0, 1) = q2 \text{ and } \delta(q5, 1) = q5$$

# Transition table of Minimized DFA

States (q)	$\delta(q_i, 0)$	$\delta(q_i, 1)$
$\longrightarrow \{q_0, q_1\}$	$\{q_0, q_1\}$	$*\{q_2, q_3, q_4\}$
$*\{q_2, q_3, q_4\}$	$*\{q_2, q_3, q_4\}$	$q_5$
$q_5$	$q_5$	$q_5$

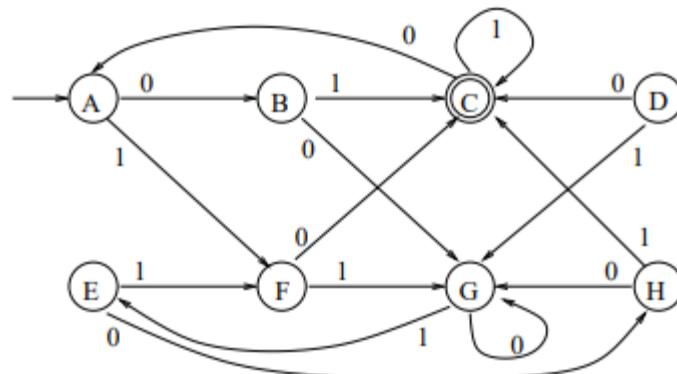
# Transition Diagram of Minimized DFA

States ( $q$ )	$\delta(q_i, 0)$	$\delta(q_i, 1)$
$\rightarrow \{q_0, q_1\}$	$\{q_0, q_1\}$	$^*\{q_2, q_3, q_4\}$
$^*\{q_2, q_3, q_4\}$	$^*\{q_2, q_3, q_4\}$	$q_5$
$q_5$	$q_5$	$q_5$



# Practice problem

1. Minimize the DFA using set partitioning method



2. Minimize the DFA using set partitioning method whose transition table is:

State	<i>Input</i>	
	a	b
$\rightarrow q_0$	$q_0$	$q_3$
$q_1$	$q_2$	$q_5$
$q_2$	$q_3$	$q_4$
$q_3$	$q_0$	$q_5$
$q_4$	$q_0$	$q_6$
$q_5$	$q_1$	$q_4$
$q_6$	$q_1$	$q_3$

## Suggested readings

1. An introduction to FORMAL LANGUAGES and AUTOMATA by PETER LINZ.
2. Introduction to Automata Theory, Languages, And Computation by JOHN E. HOPCROFT, RAJEEV MOTWANI, JEFFREY D. ULLMAN
3. Theory of computer science: automata, languages and computation by K.L.P MISHRA

# Thank you

# Theory of Computation: CS-202

## Deterministic Finite Automata Minimization

# Outlines

## Deterministic Finite Accepters (DFA)

- Minimization of DFA

- Equivalence Theorem or Set Partitioning method
- Myhill- Nerode Theorem or Table filling method

# Minimization of DFA

- Minimization of DFA means reducing the number of states from the given DFA.

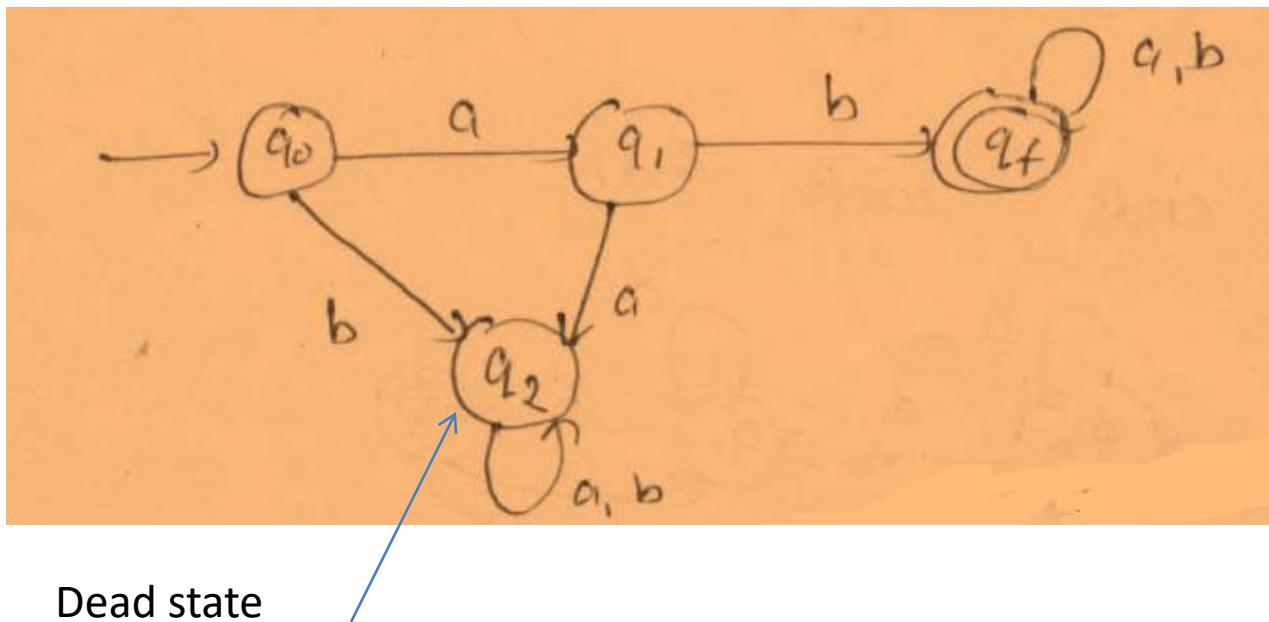
# Minimization of DFA

DFA minimization is possible by considering following states:

1. Dead states
2. Unreachable states
3. Equal or indistinguishable states

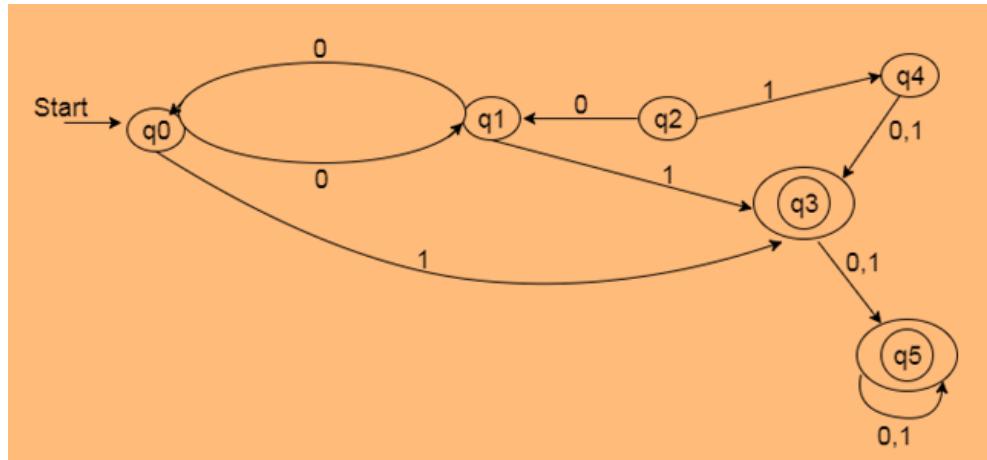
# Minimization of DFA

- **Dead states:** A dead state is non accepting state whose transitions for every input symbols terminate on themselves.
- There is no way for dead state to reach a final state.
- Merge all the dead states of DFA.



# Minimization of DFA

- **Unreachable states:** Unreachable states are the states that are not reachable from the initial state of the DFA for any input string.
- Remove unreachable states from DFA.



q2 and q4 are unreachable

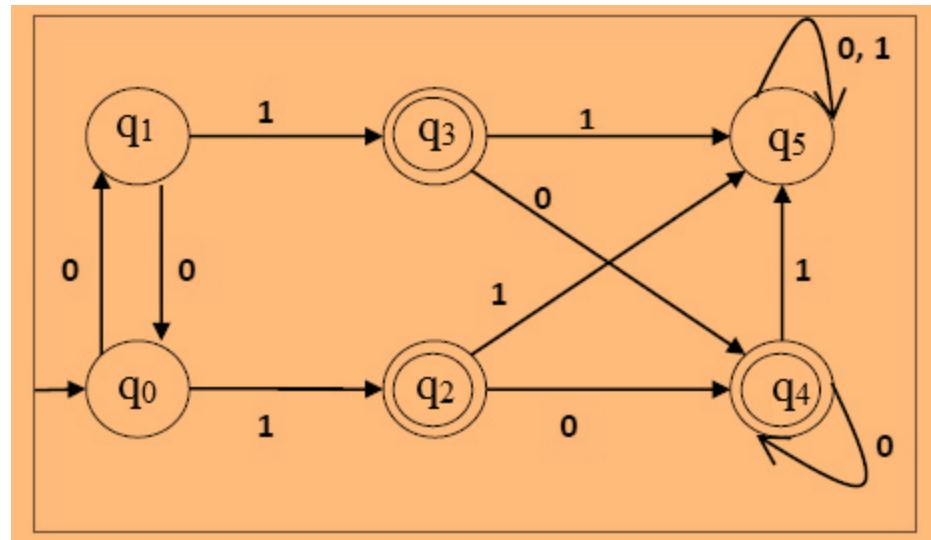
# Minimization of DFA using Myhill-Nerode Theorem or Table filling Method

Minimization of DFA means reducing the number of states from given FA.

We have to follow the various steps to minimize the DFA. These are as follows:

- **Step 1** – Draw a table for all pairs of states  $(q_i, q_j)$  not necessarily connected directly [All are unmarked initially]
- **Step 2** – Consider every state pair  $(q_i, q_j)$  in the DFA where  $q_i \in F$  and  $q_j \notin F$  or vice versa and mark them. [Here  $F$  is the set of final states]
- **Step 3** – If there is an unmarked pair  $(q_i, q_j)$ , mark it if the pair  $\{\delta(q_i, X), \delta(q_j, X)\}$  is marked for some input alphabet.
- Repeat this step until we cannot mark anymore states
- **Step 4** – Combine all the unmarked pair  $(q_i, q_j)$  and make them a single state in the reduced DFA.

# Example: Minimize the DFA using Myhill-Nerode Theorem or Table filling Method



Input string: 10000

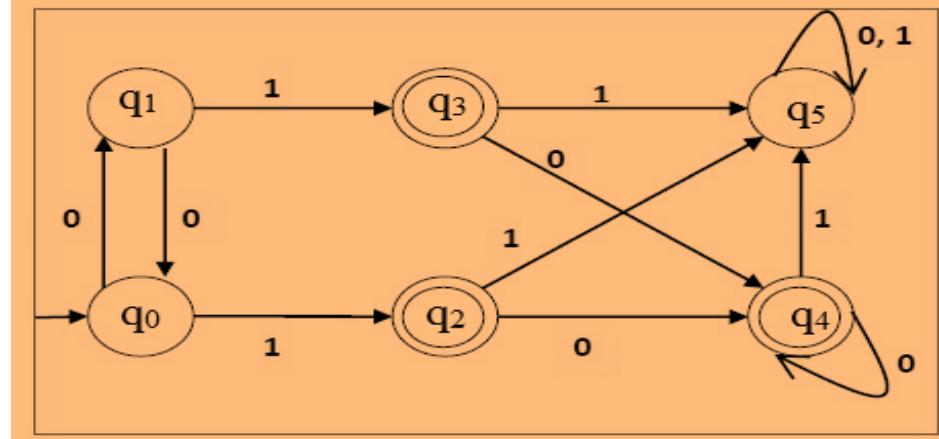
**Step 1** – Draw a table for all pair of states.

	q0	q1	q2	q3	q4	q5
q0						
q1						
q2						
q3						
q4						
q5						

**Step 2** – We mark the pairs of state .

	q0	q1	*q2	*q3	*q4	q5
q0						
q1						
*q2	✓	✓				
*q3	✓	✓				
*q4	✓	✓				
q5			✓	✓	✓	

	q0	q1	*q2	*q3	*q4	q5
q0						
q1						
*q2	✓	✓				
*q3	✓	✓				
*q4	✓	✓				
q5			✓	✓	✓	



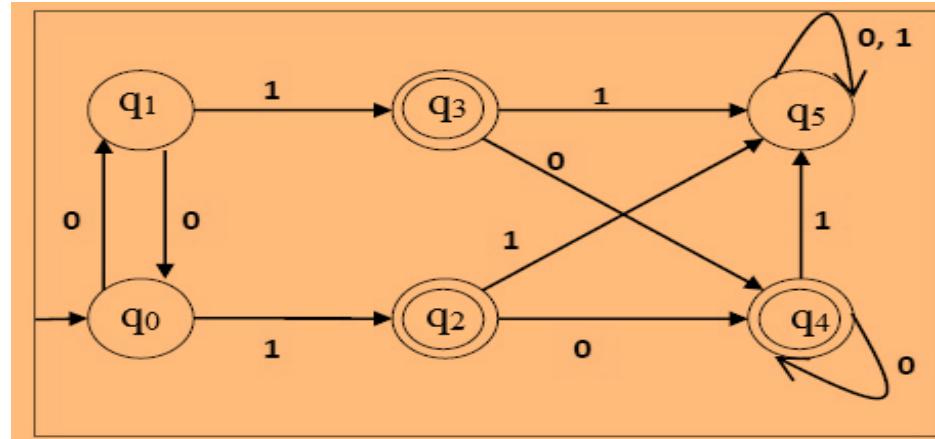
**Step 3- consider the unmarked pair {q0, q1}.**

$$\{ \delta(q0,0), \delta(q1,0) \} = \{q1, q0\}$$

$$\{ \delta(q0,1), \delta(q1,1) \} = \{q2, q3\},$$

As neither {q1, q0} nor {q2, q3} are not marked in the table, hence {q0, q1} will remain unmarked

	q0	q1	*q2	*q3	*q4	q5
q0						
q1						
*q2	✓	✓				
*q3	✓	✓				
*q4	✓	✓				
q5			✓	✓	✓	



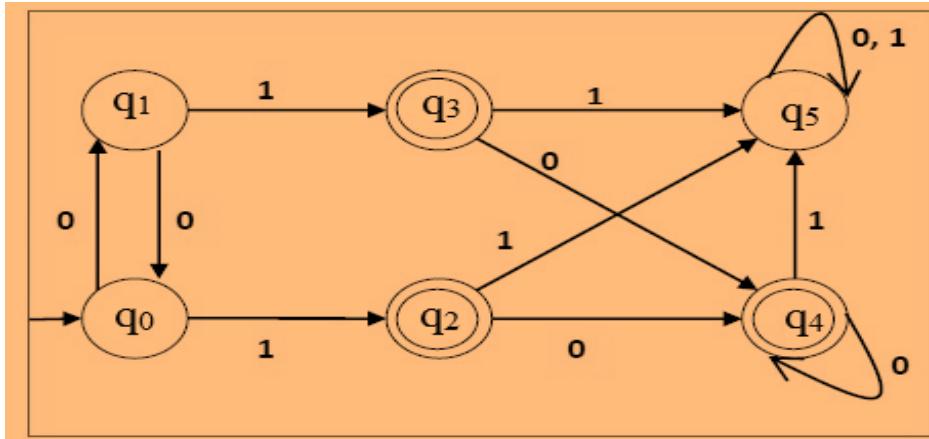
**Step 3- consider the unmarked pair {q3, q2}.**

$$\{ \delta(q3,0), \delta(q2,0) \} = \{q4, q4\}$$

$$\{ \delta(q3,1), \delta(q2,1) \} = \{q5, q5\},$$

As neither {q4, q4} nor {q5, q5} are present in the table, hence {q3, q2} will remain unmarked

	q0	q1	*q2	*q3	*q4	q5
q0						
q1						
*q2	✓	✓				
*q3	✓	✓				
*q4	✓	✓				
q5			✓	✓	✓	



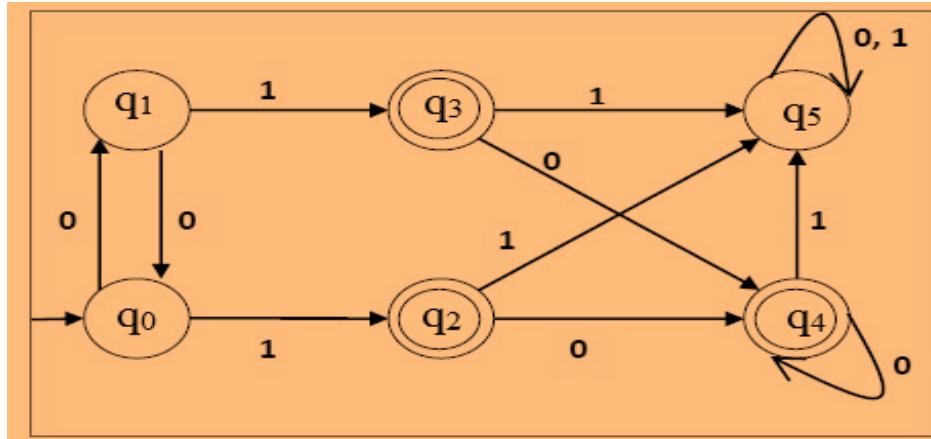
**Step 3- consider the unmarked pair {q4, q2}.**

$$\{ \delta(q4,0), \delta(q2,0) \} = \{q4, q4\}$$

$$\{ \delta(q4,1), \delta(q2,1) \} = \{q5, q5\},$$

As neither {q4, q4} nor {q5, q5} are present in the table, hence {q4, q2} will remain unmarked

	q0	q1	*q2	*q3	*q4	q5
q0						
q1						
*q2	✓	✓				
*q3	✓	✓				
*q4	✓	✓				
q5			✓	✓	✓	



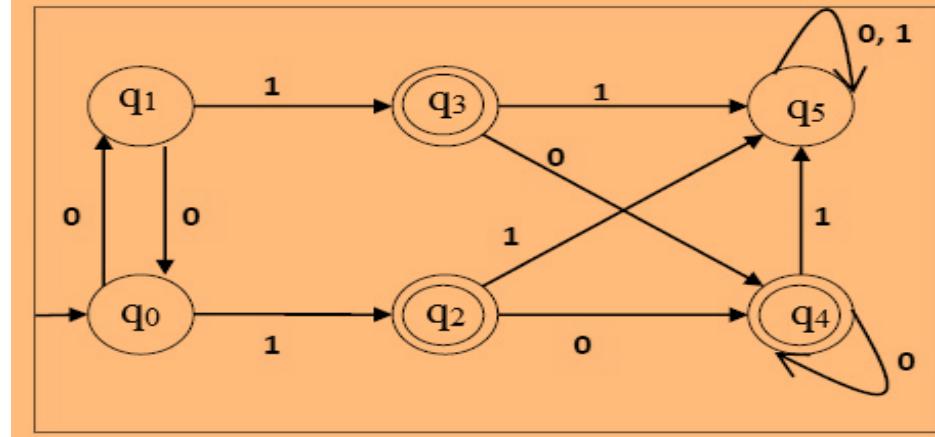
**Step 3- consider the unmarked pair {q4, q3}.**

$$\{ \delta(q4,0), \delta(q3,0) \} = \{q4, q4\}$$

$$\{ \delta(q4,1), \delta(q3,1) \} = \{q5, q5\},$$

As neither {q4, q4} nor {q5, q5} are present in the table, hence {q4, q2} will remain unmarked

	q0	q1	*q2	*q3	*q4	q5
q0						
q1						
*q2	✓	✓				
*q3	✓	✓				
*q4	✓	✓				
q5			✓	✓	✓	



**Step 3- consider the unmarked pair {q5, q0} or {q0,q5}.**

$$\{ \delta(q0,0), \delta(q5,0) \} = \{q1, q5\}$$

$$\{ \delta(q0,1), \delta(q5,1) \} = \{q2, q5\}$$

{q2, q5} is already marked, hence we will mark pair {q0, q5}.

**consider the unmarked pair {q5, q1} or {q1,q5}.**

$$\{ \delta(q1,0), \delta(q5,0) \} = \{q0, q5\}$$

$$\{ \delta(q1,1), \delta(q5,1) \} = \{q3, q5\}$$

{q3, q5} is already marked, hence we will mark pair {q1, q5}.

	q0	q1	q2	q3	q4	q5
q0						
q1						
*q2	✓	✓				
*q3	✓	✓				
*q4	✓	✓				
q5	✓ /	✓ /	✓	✓	✓	

**Step 4-** After step 3, we have got state combinations  $\{q_0, q_1\}$   $\{q_2, q_3\}$   $\{q_2, q_4\}$   $\{q_3, q_4\}$  that are unmarked.

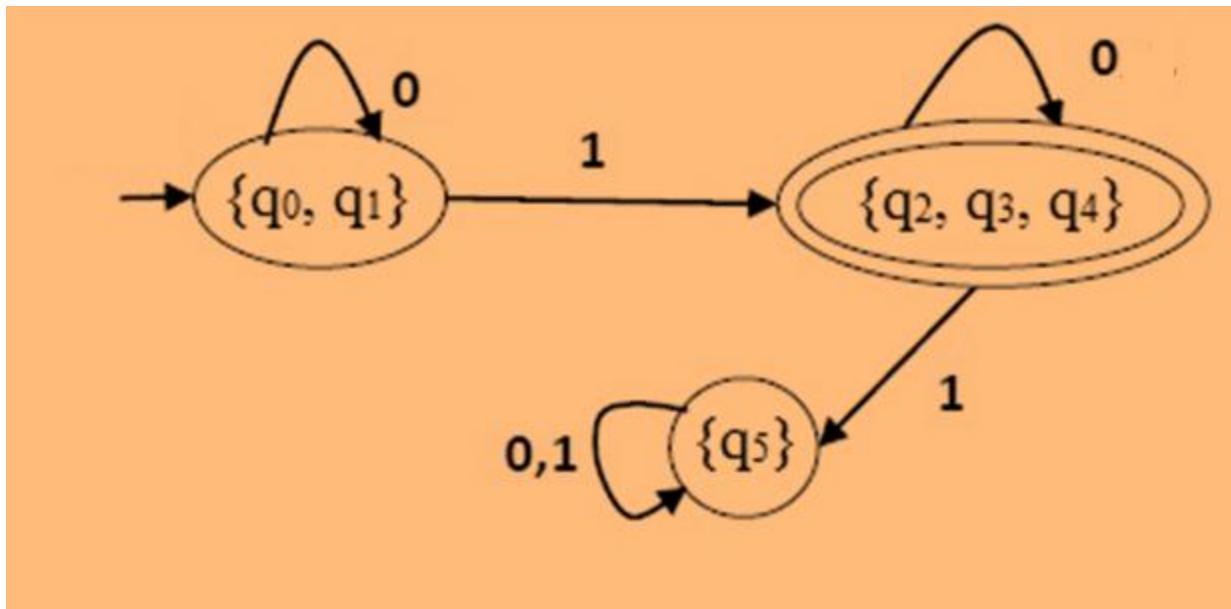
We can recombine  $\{q_2, q_3\}$   $\{q_2, q_4\}$   $\{q_3, q_4\}$  into  $\{q_2, q_3, q_4\}$

Hence we got two combined states as  $\{q_0, q_1\}$  and  $\{q_2, q_3, q_4\}$

So,  $Q = \{\{q_0, q_1\}, \{q_2, q_3, q_4\}, \{q_5\}\}$

# Transition Diagram of Minimized DFA

States ( $q$ )	$\delta(q_i, 0)$	$\delta(q_i, 1)$
$\rightarrow \{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_2, q_3, q_4\}$
$*\{q_2, q_3, q_4\}$	$*\{q_2, q_3, q_4\}$	$q_5$
$q_5$	$q_5$	$q_5$

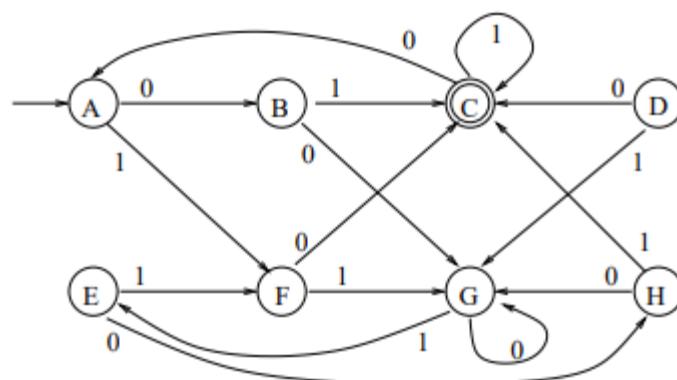


# Language accepting capabilities

- The minimized DFA will be equivalent to the DFA.
- DFA and minimized DFA accepts the same language.
- The minimized DFA will be unique.
- Only the number of states will be reduced in the minimized DFA.
- The Power of NFA and DFA is also same.

# Practice problem

1. Minimize the DFA using Table filling Method



2. Minimize the DFA using Table filling Method  
whose transition table is:

State	<i>Input</i>	
	a	b
$\rightarrow q_0$	$q_0$	$q_3$
$q_1$	$q_2$	$q_5$
$q_2$	$q_3$	$q_4$
$q_3$	$q_0$	$q_5$
$q_4$	$q_0$	$q_6$
$q_5$	$q_1$	$q_4$
$q_6$	$q_1$	$q_3$

## Suggested readings

1. An introduction to FORMAL LANGUAGES and AUTOMATA by PETER LINZ.
2. Introduction to Automata Theory, Languages, And Computation by JOHN E. HOPCROFT, RAJEEV MOTWANI, JEFFREY D. ULLMAN
3. Theory of computer science: automata, languages and computation by K.L.P MISHRA

# Thank you

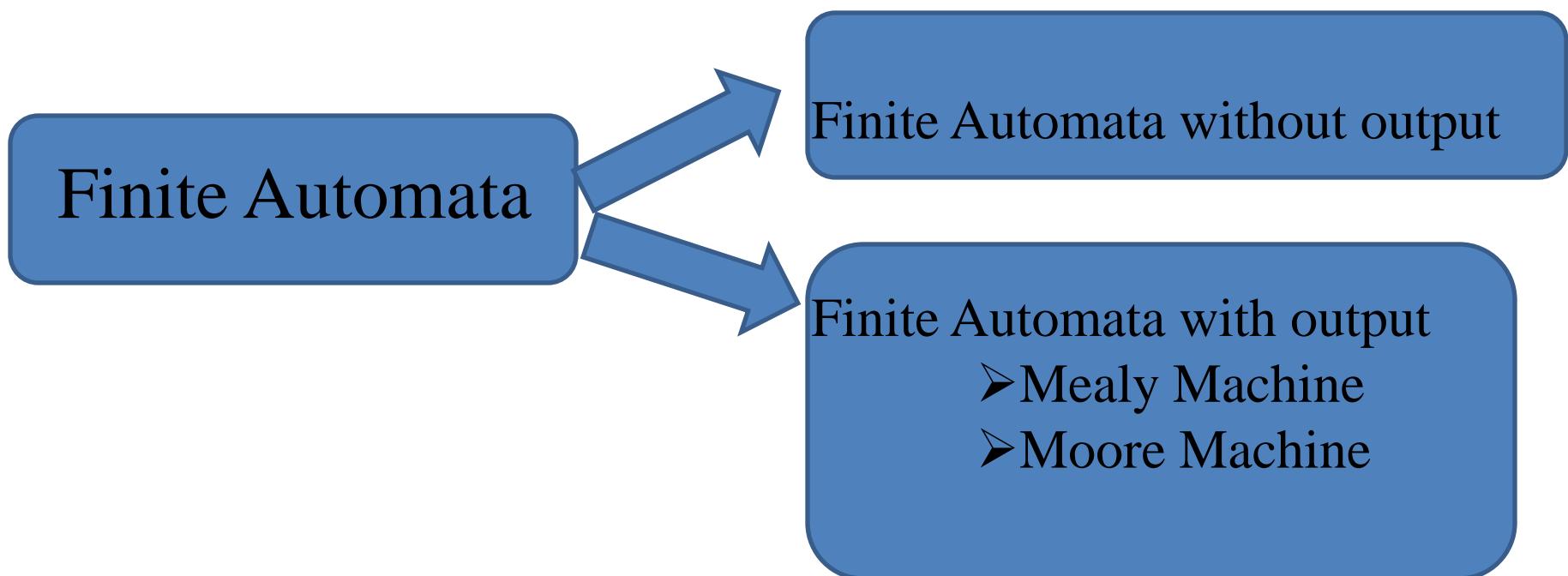
# Theory of Computation: CS-202

## Finite Automata with Output

# Outlines

- Finite Automata without Output
- Finite Automata with Output
  - Mealy Machine
  - Moore Machine
  - Conversion from Mealy to Moore Machine
  - Conversion from Moore to Mealy Machine

# Finite Automata



# Mealy Machine

A Mealy Machine is an FA whose output depends on the present state as well as the present input.

In Mealy machine every transition for a particular input symbol has a fixed output.

In the Mealy machine, the output is represented with each input symbol for each state and separated by /.

# Mealy Machine

The Mealy machine can be described by 6 tuples

$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$  where

$Q$ : finite set of states

$q_0$ : initial state of machine

$\Sigma$ : finite set of input alphabet

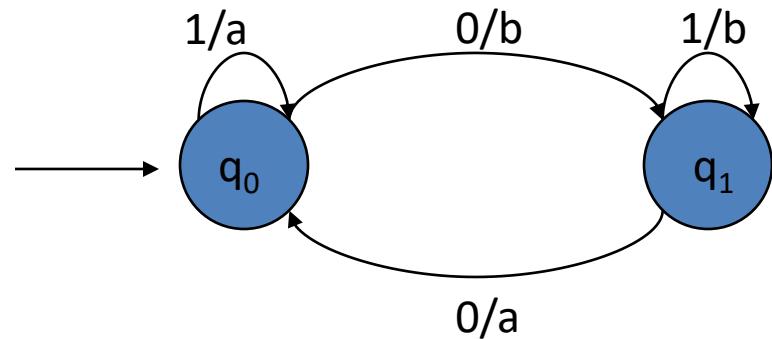
$\Delta$  : output alphabet

$\delta$ : transition function where  $Q \times \Sigma \rightarrow Q$

$\lambda$ : output function where  $Q \times \Sigma \rightarrow \Delta$

# Mealy Machine

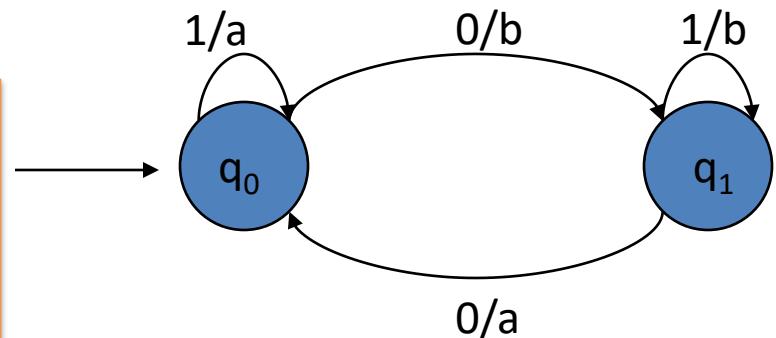
$$\Delta = \{a, b\}$$
$$\Sigma = \{0, 1\}$$



state	Input 0		Input 1	
	State	Output	State	Output
$q_0$	$q_1$	b	$q_0$	a
$q_1$	$q_0$	a	$q_1$	b

# Mealy Machine

state	Input 0		Input 1	
	State	Output	State	Output
q0	q1	b	q0	a
q1	q0	a	q1	b



Let us take an example: 110

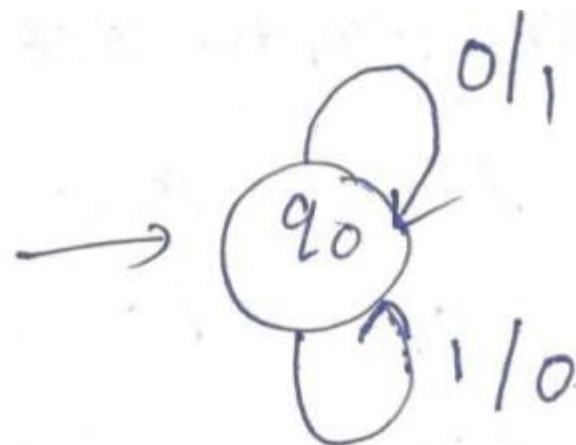
$$\delta(q_0, 1) \xrightarrow{\quad} \delta(q_0, 1) \xrightarrow{\quad} \delta(q_0, 0) \xrightarrow{\quad} q_1$$

output a                  a                  b

# Example of Mealy Machine

Design a Mealy Machine that produces the 1's complement of the given input binary string.

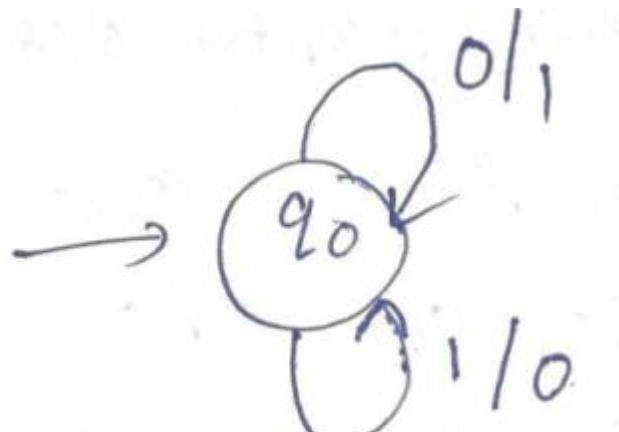
Consider the Mealy machine with  $\Sigma=\{0,1\}$  and  $\Delta=\{0, 1\}$



# Example of Mealy Machine

Solution (cont..)

Consider the Mealy machine with  $\Sigma = \{0,1\}$  and  $\Delta = \{0, 1\}$



Let us take an example: 110

Transition:  $\delta(q_0, 1) \rightarrow \delta(q_0, 1) \rightarrow \delta(q_0, 0) \rightarrow q_0$   
output: 0 0 1

# Moore Machine

A Moore Machine is an FA whose output depends on the current state only.

# Moore Machine

A Moore Machine is FA whose output depends on the present state.

The Moore machine can be described by 6 tuples

$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$  where

$Q$ : finite set of states

$q_0$ : initial state of machine

$\Sigma$ : finite set of input alphabet

$\Delta$  : output alphabet

$\delta$ : transition function where  $Q \times \Sigma \rightarrow Q$

$\lambda$ : output function where  $Q \rightarrow \Delta$

# Example of Moore Machine

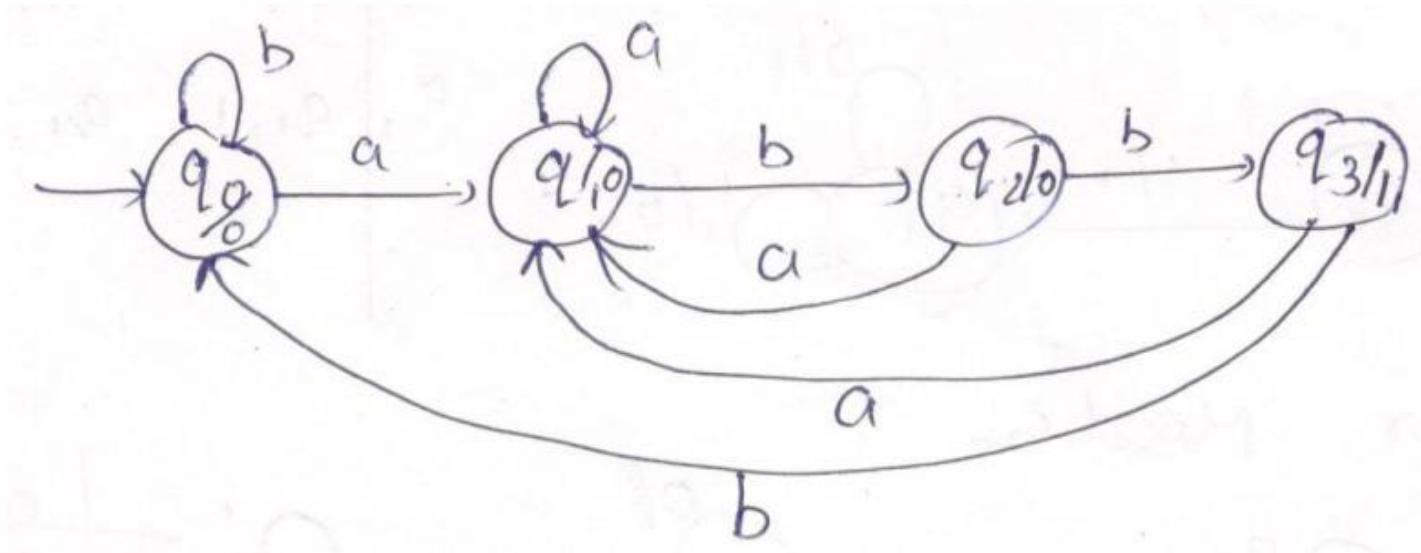
Design a Moore Machine that counts the occurrence of ‘abb’ in any input string.

Consider the Mealy machine with  $\Sigma=\{a, b\}$  and  $\Delta=\{0, 1\}$

# Example of Moore Machine

Solution (cont..)

Consider the Moore machine with  $\Sigma = \{a, b\}$  and  $\Delta = \{0, 1\}$



Let us take an example: ababb

Transition:  $\delta(q0,a) \rightarrow \delta(q1,b) \rightarrow \delta(q2,a) \rightarrow \delta(q1,b) \rightarrow \delta(q2,b) \rightarrow q3$   
output: 0 0 0 0 1

# Conversion from Mealy machine to Moore Machine

# Conversion from Mealy machine to Moore Machine

**Step 1.** First find out those states which have more than one outputs associated with them.

**Step 2.** Create two states for these states.

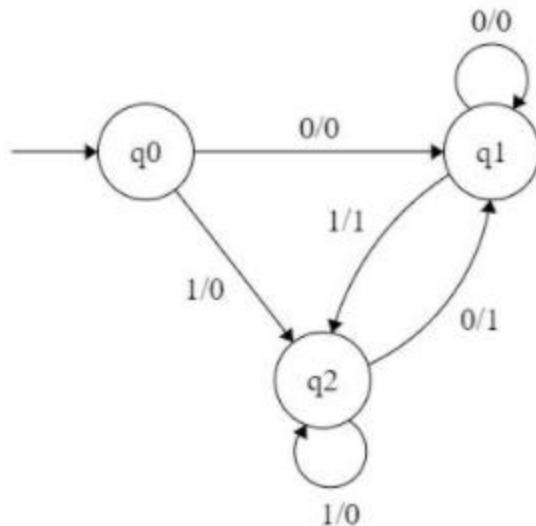
**Step 3.** Create an empty moore machine table with new generated state.

For moore machine, Output will be associated to each state.

**Step 4.** Fill the entries of next state using mealy machine transition table .

# Example

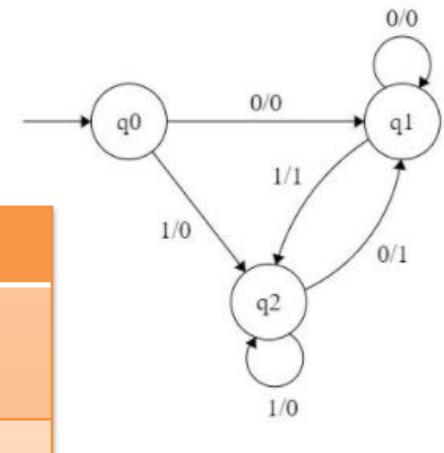
Consider the Mealy machine with  $\Sigma=\{0,1\}$  and  $\Delta=\{0, 1\}$



# Conversion from Mealy machine to Moore Machine

Consider the Mealy machine with  $\Sigma=\{0,1\}$  and  $\Delta=\{0, 1\}$

Input 0		Input 1		
Present States	Next State	Output	Next State	output
q0	q1	0	q2	0
q1	q1	0	q2	1
q2	q1	1	q2	0



# Conversion from Mealy machine to Moore Machine

**Step 1.** First find out those states which have more than one outputs associated with them. Here,  $q_1$  and  $q_2$  are the states which have both output 0 and 1 associated with them.

**Step 2.** Create two states for these states.

For  $q_1$ , two states will be  $q_{10}$  (state with output 0) and  $q_{11}$  (state with output 1). Similarly for  $q_2$ , two states will be  $q_{20}$  and  $q_{21}$ .

**Step 3.** Create an empty moore machine table with new generated state.

For moore machine, Output will be associated to each state.

**Step 4.** Fill the entries of next state using mealy machine transition table .

For  $q_0$  on input 0, next.state is  $q_{10}$  ( $q_1$  with output 0). Similarly, for  $q_0$  on input 1, next state is  $q_{20}$  ( $q_2$  with output 0).

For  $q_1$  (both  $q_{10}$  and  $q_{11}$ ) on input 0, next state is  $q_{10}$ . Similarly, for  $q_1$ (both  $q_{10}$  and  $q_{11}$ ) on input 1, next state is  $q_{21}$ .

For  $q_{10}$ , output will be 0 and for  $q_{11}$ , output will be 1. Similarly, other entries for  $q_{20}$  and  $q_{21}$  can be filled.

# Conversion from Mealy machine to Moore Machine

Input 0		Input 1	
Present States	Next State	Next State	output
q0	q10	q20	0
q10	q10	q21	0
q11	q10	q21	1
q20	q11	q20	0
q21	q11	q20	1

# Conversion from Moore machine to Mealy Machine

# Conversion from Moore machine to Mealy Machine

**Step 1.** Construct an empty mealy machine table using all states of Moore machine.

**Step 2.** Next state for each state can also be directly found from Moore machine transition Table

**Step 3.** As we can see output corresponding to each input in moore machine transition table. Use this to fill the Output entries. e.g.; Output corresponding to q10, q11, q20 and q21.

**Step 4.**

As we can see from table 6, q10 and q11 are similar to each other (same value of next state and Output for different Input). Similarly, q20 and q21 are also similar. So, q11 and q21 can be eliminated.

# Example

Convert the Moore machine to Mealy Machine

	Input 0	Input 1	
Present States	Next State	Next State	output
q0	q10	q20	0
q10	q10	q21	0
q11	q10	q21	1
q20	q11	q20	0
q21	q11	q20	1

# Conversion from Moore machine to Mealy Machine

Step 1

		Input 0	Input 1	
Present States	Next State	Output	Next State	output
q0				
q10				
q11				
q20				
q21				

# Conversion from Moore machine to Mealy Machine

Step 2

Input 0		Input 1		
Present States	Next State	Output	Next State	output
q0	q10		q20	
q10	q10		q21	
q11	q10		q21	
q20	q11		q20	
q21	q11		q20	

# Conversion from Moore machine to Mealy Machine

Step 3

Mealy machine table

Input 0		Input 1		
Present States	Next State	output	Next State	output
q0	q10	0	q20	0
q10	q10	0	q21	1
q11	q10	0	q21	1
q20	q11	1	q20	0
q21	q11	1	q20	0

Moore machine table

	Input 0	Input 1	
Present States	Next State	Next State	output
q0	q10	q20	0
q10	q10	q21	0
q11	q10	q21	1
q20	q11	q20	0
q21	q11	q20	1

# Conversion from Moore machine to Mealy Machine

	Input 0		Input 1	
Present States	Next State		Next State	output
q0	q10	0	q20	0
q10	q10	0	q21	1
q11	q10	0	q21	1
q20	q11	1	q20	0
q21	q11	1	q20	0

Step 4 As we can see from step 3, q10 and q11 are similar to each other (same value of next state and Output for different Input). Similarly, q20 and q21 are also similar  
So, q11 and q21 can be eliminated .

# Conversion from Moore machine to Mealy Machine

Present States	Input 0		Input 1	
	Next State	Output	Next State	output
q0	q1	0	q20	0
q10	q10	0	q20	1
q20	q11	1	q20	0

## Suggested readings

1. An introduction to FORMAL LANGUAGES and AUTOMATA by PETER LINZ.
2. Introduction to Automata Theory, Languages, And Computation by JOHN E. HOPCROFT, RAJEEV MOTWANI, JEFFREY D. ULLMAN
3. Theory of computer science: automata, languages and computation by K.L.P MISHRA

# Thank you

# Theory of Computation: CS-202

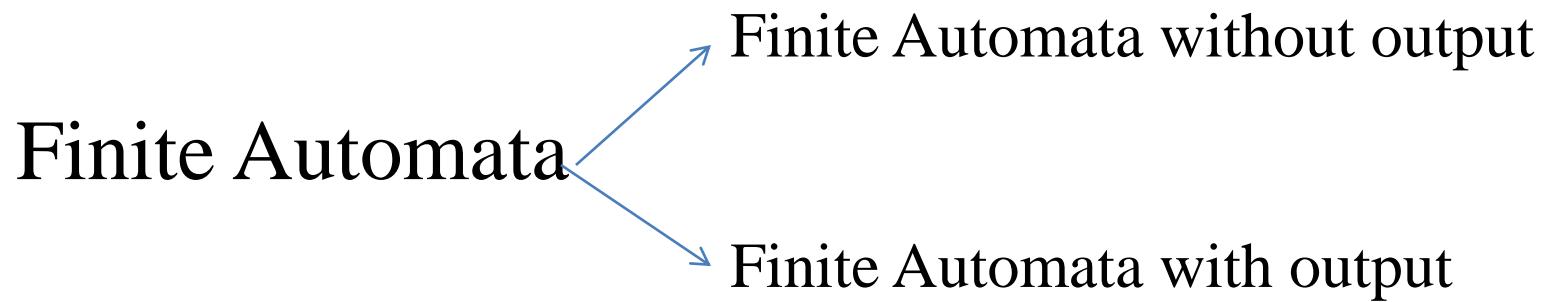
## Regular Expression

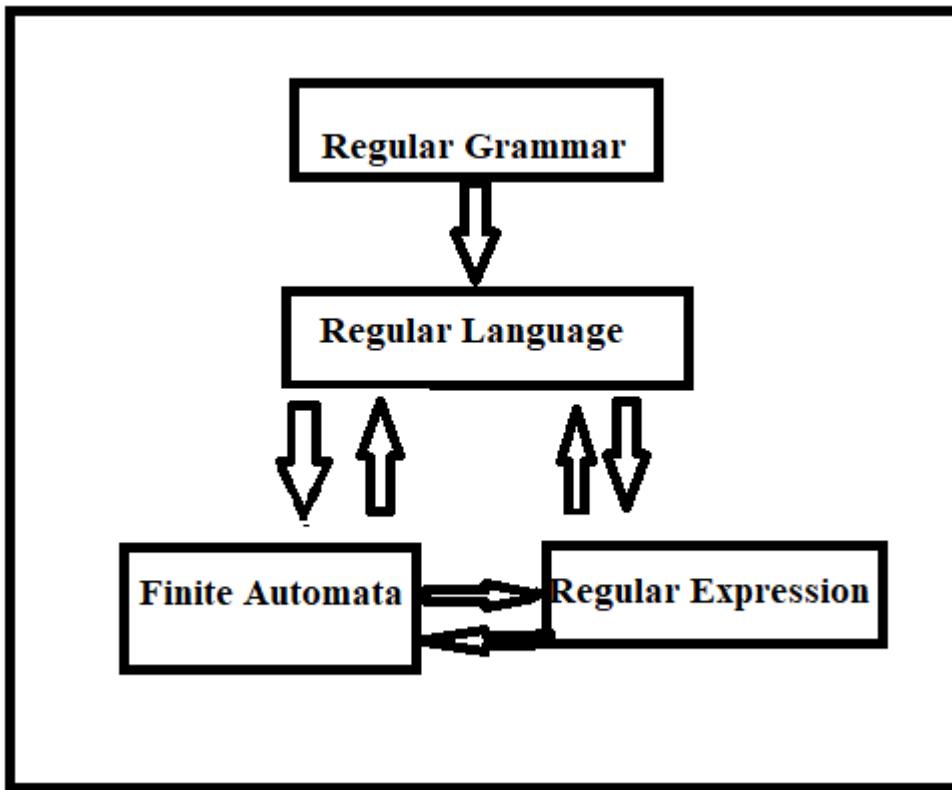
# Outline

□ Regular Expressions

□ Regular Languages

# Finite Automata





# Formal definition of Regular Expression

Let  $\Sigma$  be a given input alphabet, then

1.  $\varnothing, \lambda$  and  $a \in \Sigma$  are all regular expression, called primitive Regular Expression (R.E).
2. If  $r_1$  and  $r_2$  are R. E. then  $(r_1 + r_2)$ ,  $(r_1 . r_2)$ ,  $r_1^*$  and  $(r_1)$  are also R.E.
3. A string is a R.E. if we can derive it from the premitive R.E. by a finite number of application of the rule 2.
4. Order of operator,  $()$ ,  $*$ ,  $.$ ,  $+$

# Example

Consider the expression  $r=(a+b.c)^*$

Let  $r_1 = b$  and  $r_2 = c$ , then  $r_1 \cdot r_2$  is a regular expression.

Let  $r_3 = a$  and  $r_4 = r_1 \cdot r_2$  then  $r_3 + r_4$  is a R. E.

Let  $r_5 = (r_3 + r_4)$

$r_6 = (r_5)$

$(r_6)^*$  is also a R.E.

$\Rightarrow r$  is a R.E.

## Order and precedence of operator

1. Star closure precedes concatenation.
2. Concatenation precedes union.

# Language associated with Regular Expression

The language  $L(r)$  denoted by any regular expression  $(r)$  is defined by the following rules:

1.  $\varphi$  is a regular expression denoting empty set { }.
2.  $\lambda$  is a regular expression denoting  $\{\lambda\}$ .
3.  $\forall a \in \Sigma$  , ‘ $a$ ’ is a regular expression denoting  $\{a\}$ .  
If  $r_1$  and  $r_2$  are regular expression , then
4.  $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
5.  $L(r_1 \cdot r_2) = L(r_1) . L(r_2)$
6.  $L((r_1)) = L(r_1)$
7.  $L(r_1)^* = (L(r_1))^*$

# Find Language associated with Regular Expression $r=(a+b)^*(a+bb)$

We have,  $r=(a+b)^*(a+bb)$

$$\begin{aligned}L(r) &= L((a+b)^*(a+bb)) \\&= L((a+b)^*) L(a+bb) \\&= (L(a+b))^* L(a) \cup L(bb) \\&= (L(a) \cup L(b))^* L(a) \cup L(bb) \\&= \{a, b\}^* \{a, bb\} \\&= \{\lambda, a, b, aa, bb, ab, ba, aaa\dots\} \{a, bb\} \\&= \{a, b, aa, bb, ba, aaa\dots\dots\dots bb, abb, aabb,\dots\dots\dots\}\end{aligned}$$

So,  $L(r)$  is the set of all the strings on  $\{a, b\}$  terminated by either a or bb.

$$\begin{aligned}L(r_1 + r_2) &= L(r_1) \cup L(r_2) \\L(r_1 \cdot r_2) &= L(r_1) \cdot L(r_2) \\L((r_1))^* &= L(r_1) \\L(r_1)^* &= (L(r_1))^*\end{aligned}$$

Find Language associated with Regular Expression  
 $r=(a)^*(a+b)$

We have,  $r=(a)^*(a+b)$

$$\begin{aligned}L(r) &= L((a)^*(a+b)) \\&= L((a)^*) L(a+b) \\&= (L(a))^* L(a) \cup L(b) \\&= (L(a))^* L(a) \cup L(b) \\&= \{a\}^* \{a, b\} \\&= \{\lambda, a, aa, aaa\dots\} \{a, b\} \\&= \{a, aa, aaa\dots\dots\dots b, ab, aab,\dots\dots\dots\}\end{aligned}$$

$$L(r_1 + r_2) = L(r_1) \cup L(r_2)$$

$$L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$$

$$L((r_1)) = L(r_1)$$

$$L(r_1)^* = (L(r_1))^*$$

Find Language associated with Regular Expression  
 $r=(aa)^* (bb)^*b$

We have,  $r=(aa)^*(bb)^*b$

$$L(r) = \{ a^{2m} b^{2n+1} : m, n \geq 0 \}$$

1. Write Regular Expression for the set of strings over {0, 1} which starts with '01'

R. E.  $r= 01 (0+1)^*$

2. Write Regular Expression for the set of strings over {0, 1} which starts and ends with '0'

R. E.  $r= 0 (0+1)^* 0$

3. Write Regular Expression for the set of strings over  $\{0, 1\}$  which starts and ends with different symbol

R. E.  $r= 0(0+1)^*1 + 1(0+1)^*0$

4. Write Regular Expression for the set of strings over  $\{0, 1\}$  which starts and ends with '0'

R. E.  $r= 0(0+1)^*0$

5. Write Regular Expression for the set of strings of length 2 over  $\{0, 1\}$

R. E.  $r = (0+1)(0+1)$

6. Write Regular Expression for the set of strings of length  $\geq 2$  over  $\{0, 1\}$

R. E.  $r = (0+1)(0+1)(0+1)^*$

7. Write Regular Expression for the set of strings (w) over{a, b},  
where,  $n_a(w)=2$

R. E.     $r= b^* a b^* a b^*$

8. Write Regular Expression for the set of strings (w) over{a, b},  
where,  $n_b(w) \bmod 3=0$

R. E.     $r= a^*(a^* b a^* b a^* b a^*)^*$                        $0, 3, 6, \dots$

9. Write Regular Expression for the set of strings (w) over{a, b},  
where,  $|w| \bmod 3=0$

R. E.  $r= ((a+b)(a+b)(a+b))^*$

0, 3, 6,.....

10. Write Regular Expression for the set of strings (w) over{a, b},  
where,  $|w| \bmod 3=2$

R. E.  $r= (a+b)(a+b)((a+b)(a+b)(a+b))^*$

# Practice Problems

1. Write Regular Expression for the set of strings of length  $\leq 2$  over  $\{0, 1\}$ .
2. Write Regular Expression for the set of strings (w) over  $\{a, b\}$ , where,  $n_a(w) \bmod 3 = 1$ .
3. Write Regular Expression for the set of strings (w) over  $\{a, b\}$ , where,  $|w| \bmod 4 = 3$ .

## Suggested readings

1. An introduction to FORMAL LANGUAGES and AUTOMATA by PETER LINZ.
2. Introduction to Automata Theory, Languages, And Computation by JOHN E. HOPCROFT, RAJEEV MOTWANI, JEFFREY D. ULLMAN
3. Theory of computer science: automata, languages and computation by K.L.P MISHRA

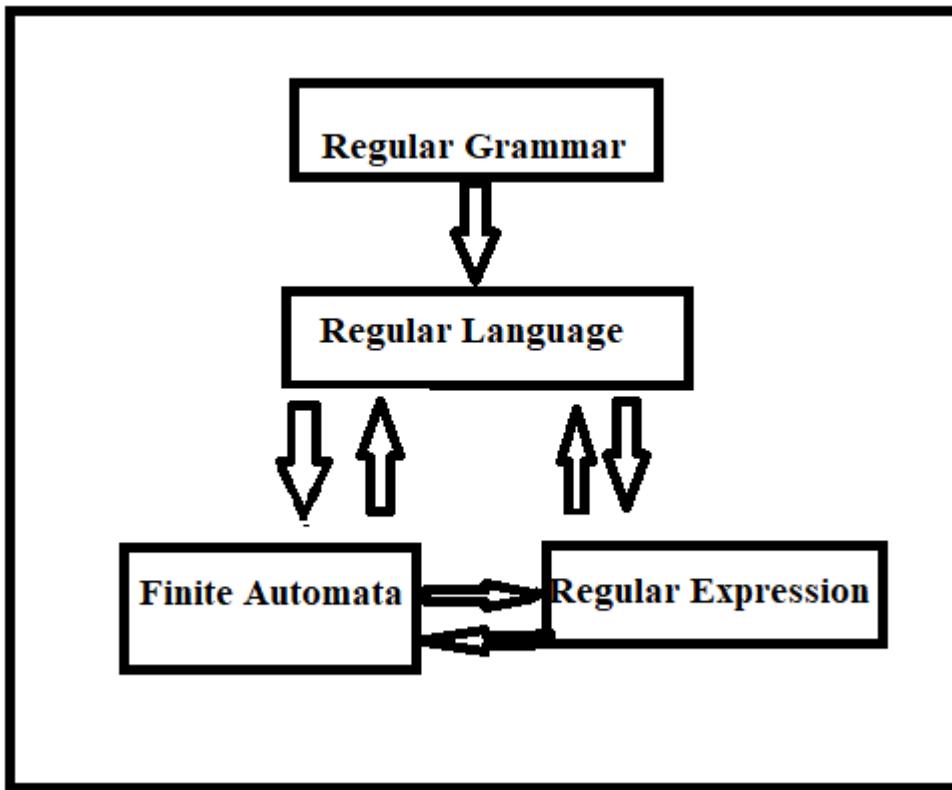
# Thank you

# Theory of Computation: CS-202

## Regular Expression

# Outline

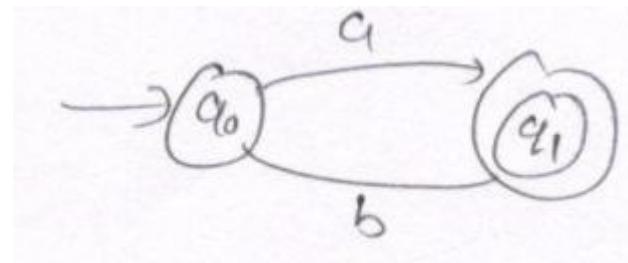
- Regular Expressions
- Regular Expression to Finite Automata
- Finite Automata to Regular Expression



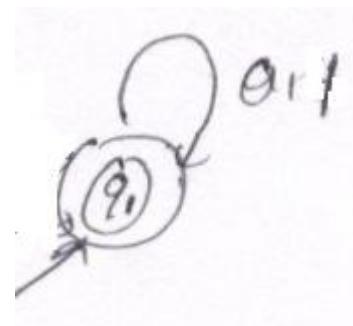
# Conversion from R.E. to finite Automata

Convert the following R.E. to finite Automata

1. R. E.       $r = (a+b)$

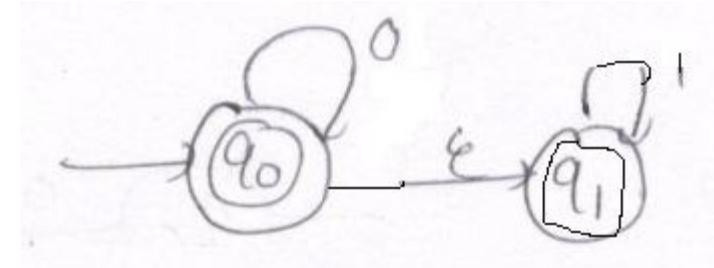


2. R. E.       $r = (0+1)^*$



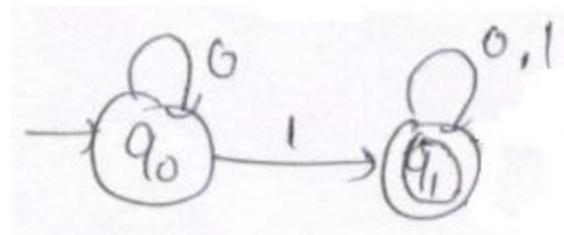
3. R. E.

$$r = 0^* 1^*$$



4. R. E.

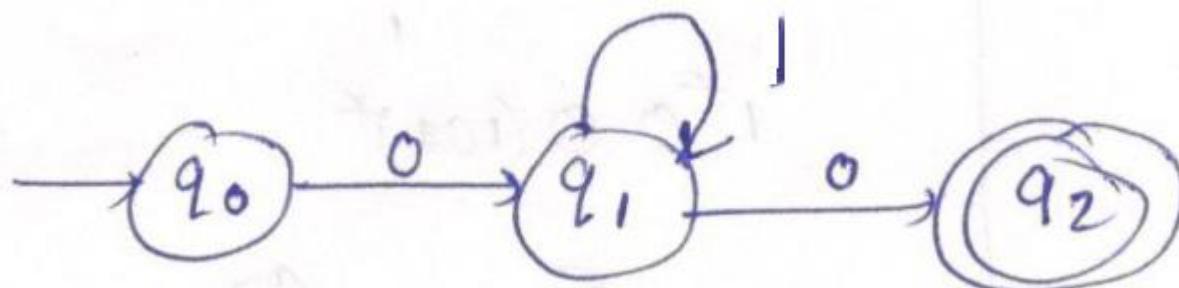
$$r = 0^* 1(0+1)^*$$



# Conversion from Finite Automata to R.E.

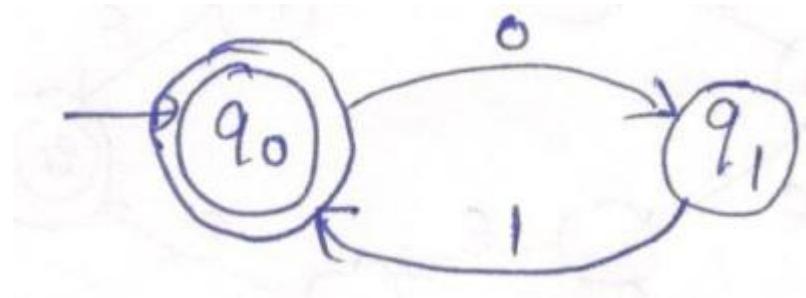
Convert the following FA to R.E.

1.



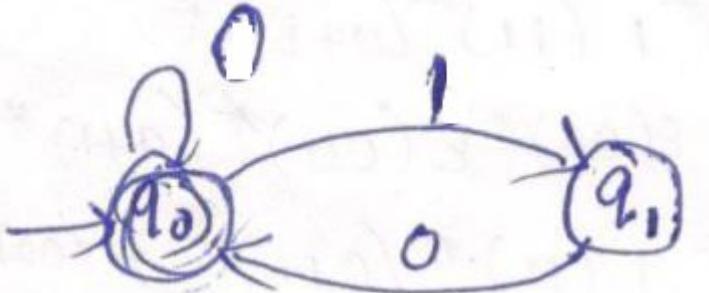
$$\text{R. E. } r = 01^*0$$

2.



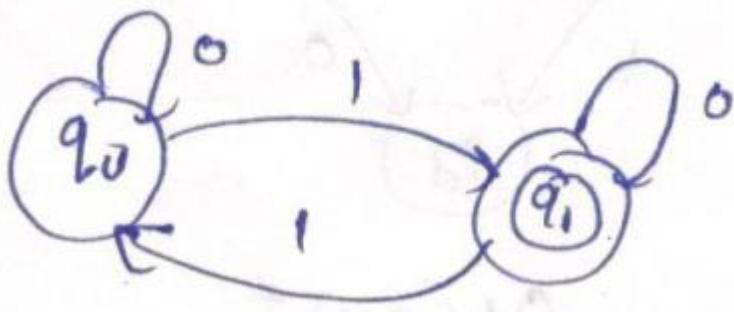
$$\text{R. E. } r = (01)^*$$

3.



R. E.  $r = (0^* + (10)^*)^*$  or  $((0+10)^*$

4.



R. E.  $r = 0^*1(0+10^*1)^*$

## Identities for Regular Expression

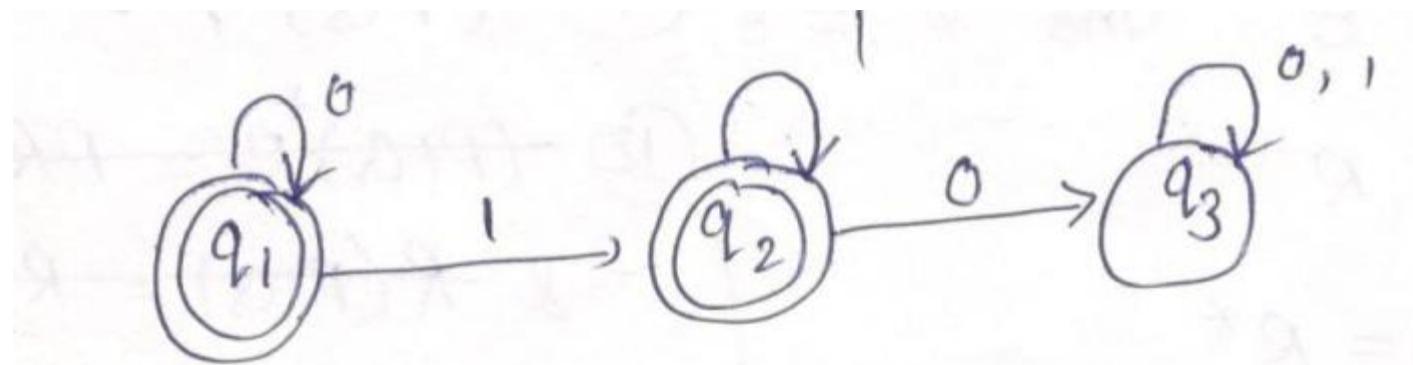
If P, Q and R are Regular Expression, then

1.  $\Phi + R = R$                                   the identity for union
2.  $\epsilon R = R \epsilon = R$                                   the identity for concatenation
3.  $\Phi R + R \Phi = \Phi$                                   the annihilator for concatenation
4.  $\epsilon^* = \epsilon$  and  $\Phi^* = \epsilon$
5.  $R + R = R$
6.  $R^* R^* = R^*$
7.  $R R^* = R^* R$
8.  $(R^*)^* = R^*$
9.  $\epsilon + R R^* = \epsilon + R^* R = R^*$
10.  $(PQ)^* P = P(QP)^*$
11.  $(P+Q)^* = (P^* Q^*) = (P^* + Q^*)^*$
12.  $(P+Q)R = PR + QR$                           &                           $R(P+Q) = RP + RQ$

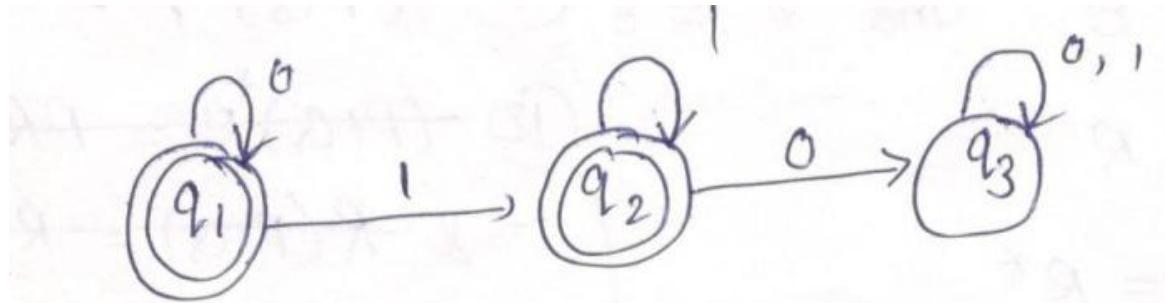
## Arden's Theorem:

Let  $P$  and  $Q$  be two R.E. over  $\Sigma$ . If  $P$  does not contain  $\epsilon$ , then the equation  $R = Q + RP$  has a unique solution  $R = QP^*$

Construct R.E. corresponding to the state diagram



Construct R.E. corresponding to the state diagram



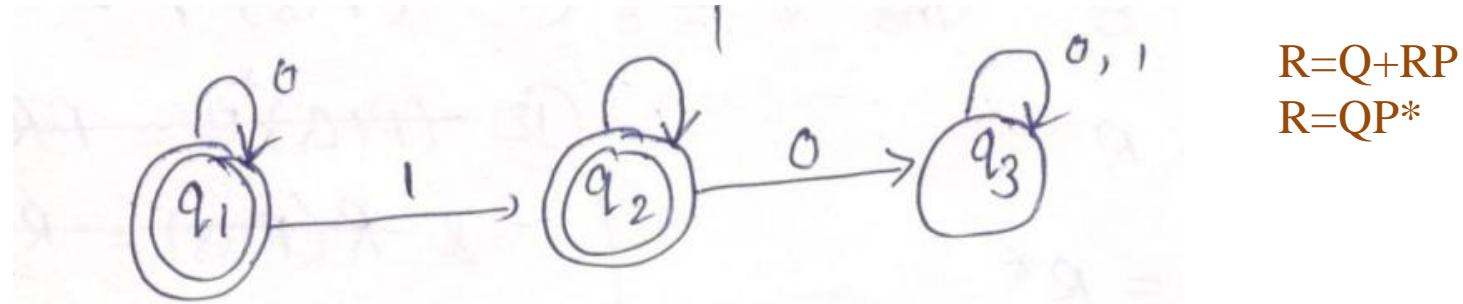
The three equations for  $q_1$ ,  $q_2$  and  $q_3$  can be written as:

$$q_1 = q_1 0 + \epsilon \dots \dots \dots (1)$$

$$q_2 = q_1 1 + q_2 1 \dots \dots \dots (2)$$

$$q_3 = q_2 0 + q_3 0 + q_3 1 \dots \dots \dots (3)$$

## Construct R.E. corresponding to the state diagram



From eq. 1

$$q_1 = q_1 0 + \epsilon \dots \dots \dots (1)$$

$$q_1 = \epsilon 0^* \quad \text{using Arden theorem}$$

From eq. 2

$$q_2 = q_1 1 + q_2 1 \dots \dots \dots (2)$$

$$q_2 = \epsilon 0^* 1 + q_2 1 = 0^* 1 + q_2 1$$

$$\Rightarrow q_2 = (0^* 1) 1^*$$

$$\text{now from eq. 3 } q_3 = q_2 0 + q_3 0 + q_3 1 \dots \dots \dots (3)$$

$$q_3 = (0^* 1) 1^* 0 + q_3 0 + q_3 1 = (0^* 1) 1^* 0 + (0+1) q_3$$

## Construct R.E. corresponding to the state diagram

$$\begin{aligned}R &= Q + RP \\R &= QP^*\end{aligned}$$

As  $q_1$  and  $q_2$  are the final states, so we need not go for state  $q_3$   
So, the require regular expression

$$R = q_1 + q_2$$

$$R = \epsilon 0^* + (0^* 1)1^*$$

$$R = 0^* + (0^* 1)1^* \quad \text{by identity 3}$$

$$R = 0^* + (0^* 1)1^*$$

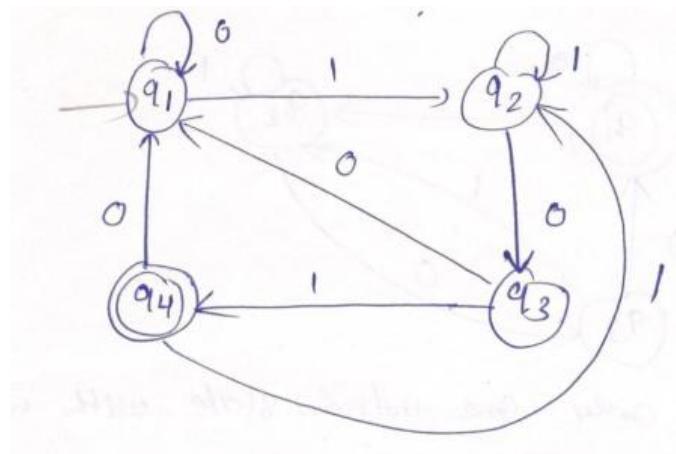
$$R = 0^*(\epsilon + 11^*)$$

$$R = 0^*(1^*) \quad \text{by identity 9}$$

$$R = 0^*1^*$$

# Practice Problems

1. Convert the R.E  $(a+bc^*d)^*$  to F.A.
2. Convert F.A to R.E using Arden's theorem.



## Suggested readings

1. An introduction to FORMAL LANGUAGES and AUTOMATA by PETER LINZ.
2. Introduction to Automata Theory, Languages, And Computation by JOHN E. HOPCROFT, RAJEEV MOTWANI, JEFFREY D. ULLMAN
3. Theory of computer science: automata, languages and computation by K.L.P MISHRA

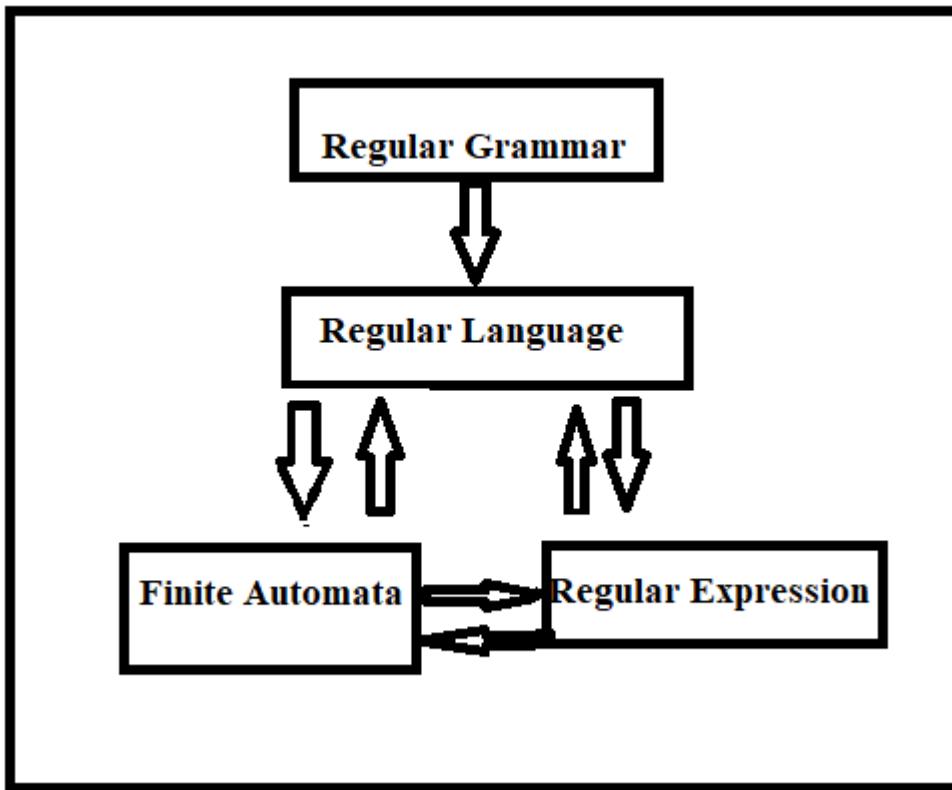
# Thank you

# Theory of Computation: CS-202

## Regular Grammar & Languages

# Outline

- Regular Grammar
- Regular Languages
- Closure properties of Regular Languages



# Regular Grammar (RG)

A third way to describe Regular Language (RL) is by means of RG.

- Right Linear Grammar
- Left Linear Grammar

# Right Linear Grammar

A Grammar  $G = (V, T, S, P)$  is said to be Right Linear if all productions are of the form

$$A \rightarrow xB$$

or

$$A \rightarrow x$$

where,  $A$  &  $B \in V$  and  $x \in T^*$

# Left Linear Grammar

A Grammar  $G = (V, T, S, P)$  is said to be Left Linear if all productions are of the form

$$A \rightarrow Bx$$

or  $A \rightarrow x$

where,  $A$  &  $B \in V$  and  $x \in T^*$

# cont...

- A Regular Grammar is either right linear or left linear

## Note:

- ❖ In RG at most one variable appear on the R.H.S. of any production and that variable must be either right most or left most in the production.
- ❖ A language  $L$  is called regular if there is a finite automata.

# Example

$$G_1 = (\{S\}, \{a, b\}, S, P_1)$$

$P_1$  given as

$$S \rightarrow abS \mid a \quad \text{is right linear.}$$

---

$$G_2 = (\{S, S_1, S_2\}, \{a, b\}, S, P_2)$$

$P_2$  is given as

$$S \rightarrow S_1ab$$

$$S_1 \rightarrow S_1ab \mid S_2$$

$$S_2 \rightarrow a \quad \text{is left linear.}$$

Both  $G_1$  &  $G_2$  are RG.

## Cont...

➤ For  $G_1$

$$S \rightarrow abS \rightarrow ababS \rightarrow aaaba$$

is a derivation

$$L(G_1) = L((ab)^*a)$$

➤ For  $G_2$

$$S \rightarrow S_1 ab \rightarrow S_1 abab \rightarrow S_2 abab \rightarrow aabab$$

R.E.  $\rightarrow a(ab)^*$

$$L(G_2) = L(a(ab)^*)$$

The  $G = (\{S, A, B\}, \{a, b\}, S, P)$

with production

$$S \rightarrow A$$

$$A \rightarrow aB \mid \lambda$$

$$B \rightarrow Ab$$

is not regular, although the production are either right linear or left linear.

- The Grammar itself is neither right linear nor left linear.
- This language is linear language

In linear language, at most one variable can occur on R.H.S. of productions without restriction on the position of this variable.

- ❖ Every RG is linear but not all linear grammar are regular.

## Pumping Lemma for Regular Language

- It is used to prove that the given language is not Regular.
- It can't be used to prove that a language is Regular.

Let  $L$  be infinite RL then there exists some positive integer  $m$  s.t. any  $W \in L$  with  $|W| \geq m$  can be decomposed as

$$W = XYZ$$

with

$$|XY| \leq m$$

$$|Y| \geq 1$$

s.t.  $W_i = XY^iZ$  is also in  $L$  for  $i = 0, 1, 2, \dots$

Note:

Pumping lemma is a negative test which can tell that the given language is not Regular, but from the definition it is not telling that if the conditions satisfy then the language will be Regular.

## Example:

$L = \{a^n b^n \mid n \geq 0\}$  is not regular, we can prove it by contradiction.

Let  $L$  is regular, then  $a^m b^m$  has a pumping length  $m$ .

➤  $L$  can be divided into  $XYZ$

$$W = aaaaabbbbb$$

take  $m$  sufficiently large.

take  $m = 5$

$Y$  can be in ‘a’ , ‘b’ or ‘ab’ part

$$\text{take } i=2, XY^2Z$$

$\underline{aaaaaabbbbb}$   $\xrightarrow{\hspace{2cm}}$   $|XY| = 5, \quad Z = 5$   
 $\underline{x} \quad \underline{y} \quad z$   $m=5$

$\underline{aaaaaabbbbb}$   $\xrightarrow{\hspace{2cm}}$   $|XY| = 8, \quad Z = 2$   $|XY| \leq m$   
 $\underline{x} \quad \underline{y} \quad \underline{z}$   $|Y| \geq 1$

$\underline{aaaaaabbbbb}$   $\xrightarrow{\hspace{2cm}}$   $|XY| = 7, \quad Z = 3$   
 $\underline{x} \quad \underline{y} \quad z$

Since, all the three conditions are not satisfied .  
 So, L is not regular.

# Closure Properties of Regular Languages

Theorem: : If  $L_1$  and  $L_2$  are R.L. then so are,  $L_1 \cup L_2$ ,  $L_1 \cap L_2$ ,  $L_1 L_2$ ,  $\overline{L_1}$  &  $L_1^*$ .

We say that the family of RL is closed under union, intersection, concatenation, complementation and star closure.

**Proof:**

If  $L_1$  and  $L_2$  are RL then there exist R.E.  $r_1$  and  $r_2$

s.t.  $L_1 = L(r_1)$  and  $L_2 = L(r_2)$

By definition,  $r_1 + r_2$ ,  $r_1 r_2$  and  $r_1^*$  are R.E. denoting languages  $L_1 \cup L_2$ ,  $L_1 L_2$  and  $L_1^*$ .

Thus, family of RL is closed under union, concatenation & star closure.

To show closure under complementation

Let  $M = (Q, \Sigma, \delta, q_0, F)$  be a DFA that accepts  $L_1$ .

then the DFA

$$\bar{M} = (Q, \Sigma, \delta, q_0, Q - F) \text{ accepts } \overline{L_1}.$$

For the given string  $w \in \Sigma^*$

$$\delta^*(q_0, w) \in F, \quad \Rightarrow w \in L_1$$

$$\delta^*(q_0, w) \in Q - F, \quad \Rightarrow w \in \overline{L_1}$$

## Closure under intersection

Let  $L_1 = L(M_1)$  and  $L_2 = L(M_2)$

Where,  $M_1 = (Q, \Sigma, \delta_1, q_0, F_1)$

$M_2 = (P, \Sigma, \delta_2, p_0, F_2)$  are DFA's.

$\Rightarrow$  We can construct from  $M_1$  and  $M_2$  a combined automaton

$$\hat{M} = (\hat{Q}, \Sigma, \hat{\delta}, (q_0, p_0), \hat{F})$$

where  $\hat{Q} = Q \times P$  consist of pairs  $(q_i, p_j)$  and transition function  $\hat{\delta}$  is such that  $\hat{M}$  is in state  $(q_i, p_j)$  whenever  $M_1$  is in state  $q_i$  &  $M_2$  is in state  $p_j$

$\Rightarrow \hat{\delta}(q_i, p_j, a) = (q_k, p_l)$

Whenever  $\delta_1(q_i, a) = q_k$

$$\delta_2(p_j, a) = p_l$$

$\hat{F}$  is defined as the set of all  $(q_i, p_j)$

s.t.  $q_i \in F_1$  &  $p_j \in F_2$

$\Rightarrow w \in L_1 \cap L_2$  iff it is accepted by  $\hat{M}$

$\Rightarrow L_1 \cap L_2$  is regular.

## Closure under other operations-

Suppose  $\Sigma$  and  $\Gamma$  are alphabets, then a  $f^n$

$$h : \Sigma \rightarrow \Gamma^*$$

is called homomorphism. Homomorphism is a substitution in which a single letter is replaced with a string.

If  $w = a_1a_2\dots a_n$

then  $h(w) = h(a_1) h(a_2)\dots h(a_n)$

If  $L$  is a language on  $\Sigma$ , then its homomorphic image is defined as

$$h(L) = \{h(w) : w \in L\}$$

## Example:

Let  $\Sigma = \{a, b\}$ ,  $\Gamma = \{a, b, c\}$  and  
h is defined by  $h(a) = ab$   
 $h(b) = bbc$

$$\begin{aligned} h(aba) &= h(a) h(b) h(a) \\ &= abbbcab \end{aligned}$$

& if  $L = \{aa, aba\}$   
homomorphic image of L is

$$h(L) = \{abab, abbbcab\}$$

# Thank you

# Theory of Computation: CS-202

## Context Free Language

# Outline

- Context Free Languages
- Context Free Grammar (CFG)
- Derivation Tree

# Context Free Language

$$L = \{a^n b^n : n \geq 0\}$$

If we take  $a = ($  &  $b = )$

Then this language describes a simple kind of nested structure found in programming.

i.e. it will accept all the strings of type  $((0))$ ,  $((0))$

but not  $(0)$

- ⇒ This shows that some properties of programming languages requires something beyond regular languages.
- ⇒ Context free language is perhaps the most important aspect of formal language theory as it applies to programming language.

# Context Free Grammar

A grammar  $G = (V, T, S, P)$  is said to be context free if all production in  $P$  have the form

$$A \rightarrow x$$

where  $A \in V$  &  $x \in (V \cup T)^*$

A language  $L$  is said to be context free iff there is a context free grammar  $G$  such that  $L = L(G)$

Note:

⇒ RG is restricted in two ways:

- left side must be a single variable
- right side has a special form

⇒ To make grammar powerful, we must relax some of these restrictions so by retaining restriction of L.H.S. but permitting anything on the right, we can get context free grammar.

# Example

➤  $G = (\{S\}, \{a, b\}, S, P)$

P :       $S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow \lambda$

is context free

$S \rightarrow aSa \rightarrow aaSaa \rightarrow aabSbaa \rightarrow aabbaa$

this makes it clear that

$$L(G) = \{ww^R : w \in (a, b)^*\}$$

Note:

Above language is CF

Note: Regular and linear grammar are C.F.

but C.F.G. is not necessarily linear.

# Leftmost and Rightmost Derivations

If CFG that is not linear, a derivation may involve sentential forms with more than one variables, so in such cases we have a choice in order by which variables are replaced.

$$G = (\{A, B, S\}, \{a, b\}, S, P)$$

P:

1.  $S \rightarrow AB$
2.  $A \rightarrow aaA$
3.  $A \rightarrow \lambda$
4.  $B \rightarrow Bb$
5.  $B \rightarrow \lambda$

$$S \xrightarrow[1]{} AB \xrightarrow[2]{} aaAB \xrightarrow[3]{} aaB \xrightarrow[4]{} aaBb \xrightarrow[5]{} aab$$

$$S \xrightarrow[1]{} AB \xrightarrow[4]{} ABb \xrightarrow[5]{} Ab \xrightarrow[2]{} aaAb \xrightarrow[3]{} aab$$

A derivation is said to be leftmost if in each step the leftmost variable in the sentential form is replaced. If in each step the rightmost variable is replaced, called rightmost derivation.

Example:

1.  $S \rightarrow aAB$
2.  $A \rightarrow aBb$
3.  $B \rightarrow A \mid \lambda$

Leftmost derivative:

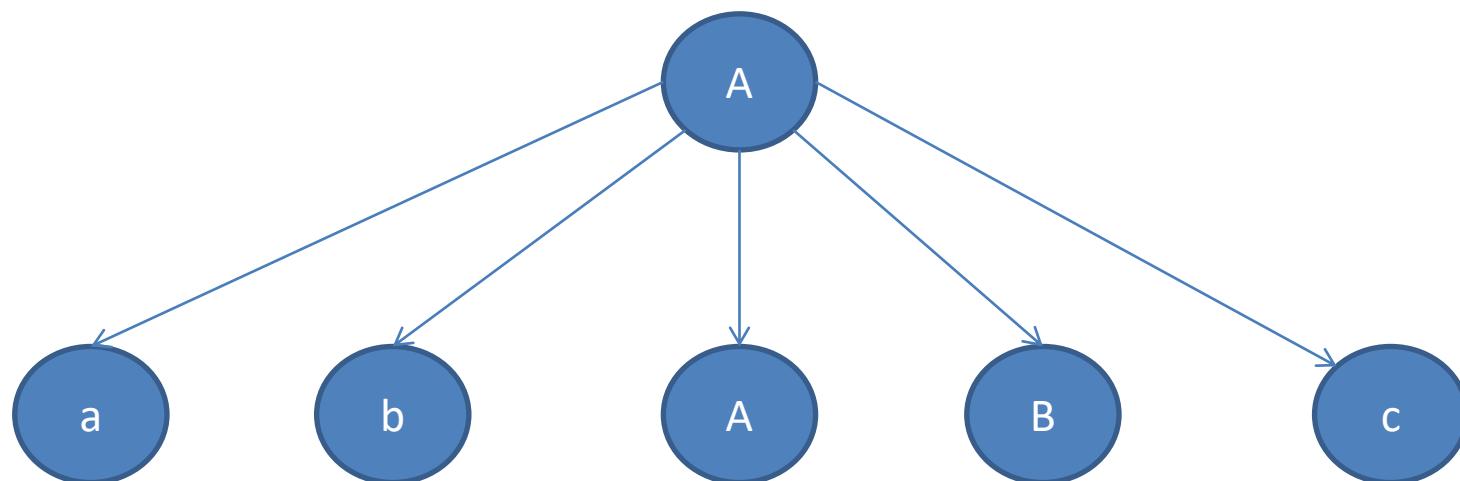
$$S \xrightarrow{1} aAB \xrightarrow{2} abBbB \xrightarrow{3} abAbB \xrightarrow{2} abbBbbB \xrightarrow{3} abbbbB \xrightarrow{3} abbbb$$

Rightmost derivative:

$$S \xrightarrow{1} aAB \xrightarrow{3} aA \xrightarrow{2} aaBb \xrightarrow{3} abAb \xrightarrow{2} abbBbb \xrightarrow{3} abbbb$$

**Derivation tree:** It is an ordered tree in which nodes are labeled with the left sides of productions and in which the children of a node represent its corresponding right side.

$$A \rightarrow abABC$$



## Definition of Derivation Tree

Let  $G = (V, T, S, P)$  be a CFG. An ordered tree is a derivation tree for G iff it has the following properties:

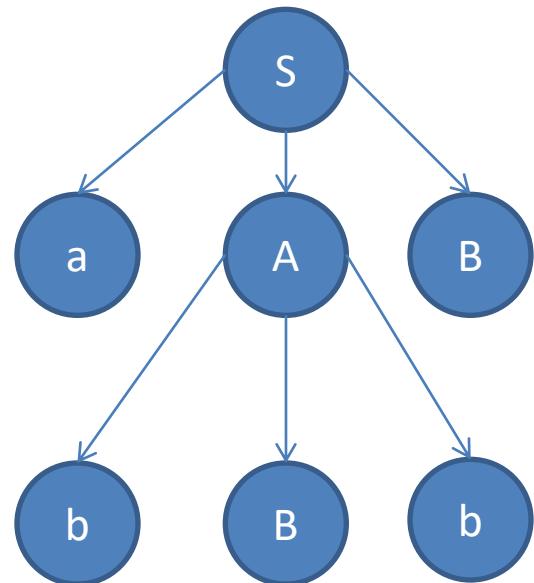
1. The root is labeled  $S$ .
2. Every leaf has a label from  $T \cup \{\lambda\}$ .
3. Every interior vertex (a vertex i.e. not leaf) has a leaf.
4. If a vertex has a label  $A \in V$  & its children are labeled (L to R)  $a_1, a_2, \dots, a_n$  then P must contain a production of form
$$A \rightarrow a_1 a_2 \dots a_n$$
5. A leaf labeled  $\lambda$  has no sibling (no children)

Note: A tree has properties 3, 4 & 5, but in which property 2 does not necessarily hold then property 2 can be replaced by 2(a).

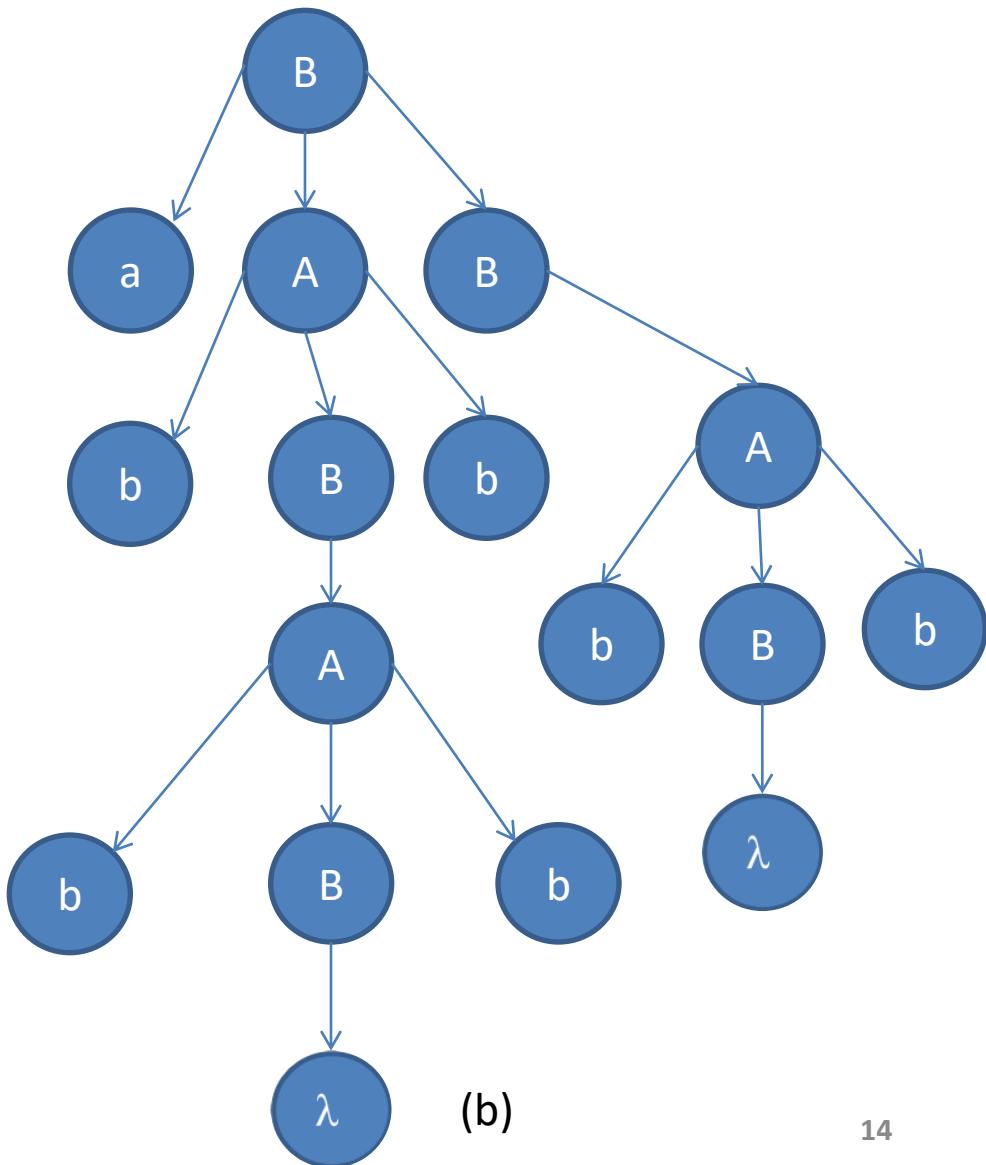
2. (a) Every leaf has a label from  $V \cup T \cup \{\lambda\}$  is said to be a partial derivation tree.

Example:

$$S \rightarrow aAB, A \rightarrow bBb, B \rightarrow A \mid \lambda$$



(a)



(b)

$\Rightarrow$  (a) is partial derivation tree for G

(b) is derivation tree

String abBbB is a sentential form of G

String abbbbb is sentence of  $L(G)$ .

# Thank you

# Theory of Computation: CS-202

## CFL: Parsing & Ambiguity

# Outlines

- Parsing
- Ambiguity
- Simplification of Context free Grammar

# Parsing

Parsing: Consider all production of the form

$$S \rightarrow x$$

Find all  $x$  that can be derived from  $S$  in one step.  
If none of these **matches** with the given string  $w$ ,  
go to the next round, in which we can apply all  
applicable production to the left most variable  
of every  $x$ .

# Example

Exhaustive search parsing

$$S \rightarrow SS \mid aSb \mid bSa \mid \lambda$$

and the string  $w = aabb$

1.  $S \rightarrow SS$
2.  $S \rightarrow aSb$
3.  $S \rightarrow bSa$
4.  $S \rightarrow \lambda$

For the given string last two productions can be removed.

1.  $S \rightarrow SS \rightarrow SSS$
2.  $S \rightarrow SS \rightarrow aSbS$
3.  $S \rightarrow SS \rightarrow bSaS$
4.  $S \rightarrow SS \rightarrow S$

*Now for sentential form*

1.  $S \rightarrow aSb \rightarrow aSSb$
2.  $S \rightarrow aSb \rightarrow aaSbb$
3.  $S \rightarrow aSb \rightarrow abSab$
4.  $S \rightarrow aSb \rightarrow ab$

*For target string, aabb*

$$S \rightarrow aSb \rightarrow aaSbb \rightarrow aabb$$

Note:

In the above equation difficulty comes from the productions  $S \rightarrow \lambda$  which is used to decrease the length of successive sentential forms.

⇒ In fact, we want to rule out, two type of productions  $A \rightarrow \lambda$  &  $A \rightarrow B$  which does not affect the power of resulting grammar in any significant way.

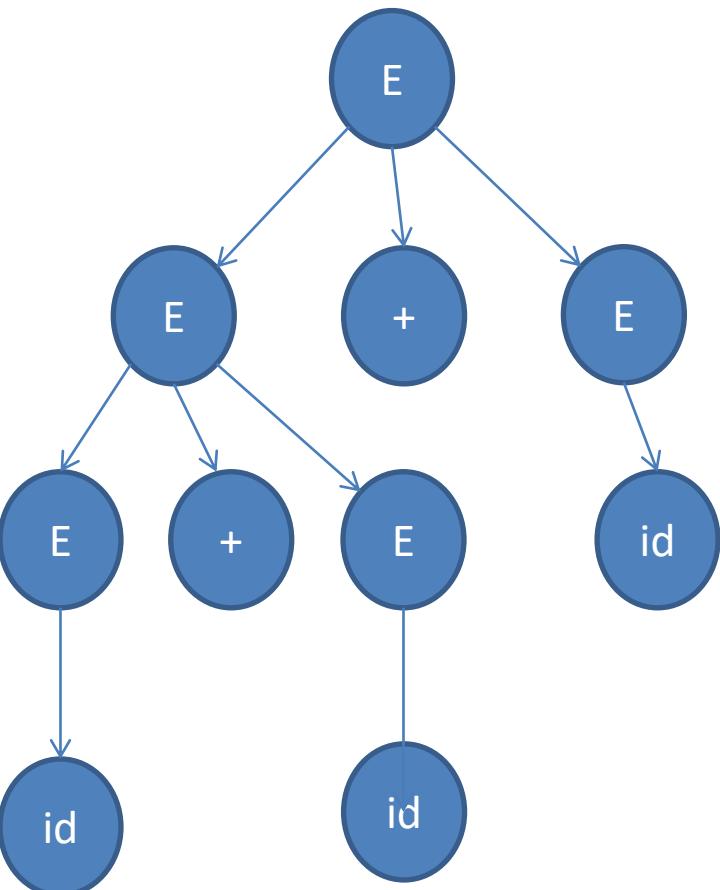
# Ambiguity in Grammar

A CFG ‘G’ is said to be ambiguous if there exist some  $w \in L(G)$  which has at least two distinct derivation trees.

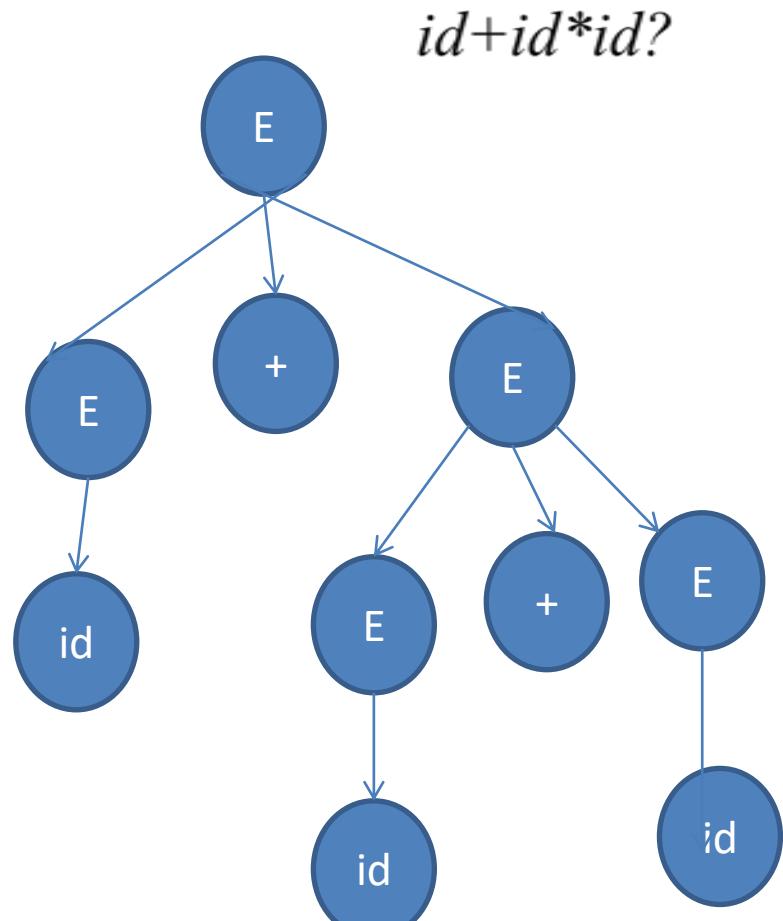
*Example:*

$$E \rightarrow E + E \mid E * E \mid id$$

string *id* + *id* + *id*



(a)



(b)

# Simplification of CFG

A useful substitution rule

Theorem: Let  $G = (V, T, S, P)$  be a context free grammar suppose that  $P$  contains a production of the form

$$\begin{aligned} A &\rightarrow x_1 B x_2 \\ B &\rightarrow y_1 \mid y_2 \mid \dots \mid y_n \end{aligned}$$

is the set of productions in  $P$  which have  $B$  as left side.

Let  $\hat{G} = (V, T, S, \hat{P})$  be the grammar in which  $\hat{P}$  is constructed by deleting  $A \rightarrow x_1 B x_2$  from  $P$  and adding to it

$$A \rightarrow x_1 y_1 x_2 \mid x_1 y_2 x_2 \mid \dots \mid x_1 y_n x_2$$

Then  $L(\hat{G}) = L(G)$

# Example

Consider  $G = (\{A, B\}, \{a, b, c\}, A, P)$

with production P:

$$A \rightarrow a \mid aaA \mid abBc$$

$$B \rightarrow abbA \mid b$$

By using substitution rule, production rules for grammar  $\hat{G}$  :

$$A \rightarrow a \mid aaA \mid ababbAc \mid abbc$$

$$L(\hat{G}) = L(G)$$

# Example

Consider  $G = (\{A, B\}, \{a, b, c\}, A, P)$

with production:

1      2      3                  4      5

$$A \rightarrow a \mid aaA \mid abBc, \quad B \rightarrow abbA \mid b$$

By using substitution rule

1      2      3                  4

$$\hat{G} = A \rightarrow a \mid aaA \mid ababbAc \mid abbc,$$

For the string aaabbc

2      3                  5

$$A \Rightarrow aaA \Rightarrow aaabBc \Rightarrow aaabbc \text{ in } G$$

2      4

$$A \Rightarrow aaA \Rightarrow aaabbc \text{ in } \hat{G}$$

# Removing useless production

Let  $G = \{V, T, S, P\}$  be a CFG. A variable  $A \in V$  is said to be useful iff there is at least one  $w \in L(G)$  s.t.

$$\begin{aligned} S &\xrightarrow{*} xAy \xrightarrow{*} w \\ x, y &\in (V \cup T)^* \end{aligned}$$

A variable that is not useful is called useless.

Example:

$$S \rightarrow A$$

$$A \rightarrow aA \mid \lambda$$

$$B \rightarrow bA$$

here variable B is useless & so the production

$$B \rightarrow bA$$

# Removing $\lambda$ - production

Any production of a CFG of the form  $A \rightarrow \lambda$  is called  $\lambda$ - production. Any variable A for which the derivation

$$A \xrightarrow{*} \lambda$$

is possible, is called null able.

Example:

1.  $S \rightarrow ABaC$
2.  $A \rightarrow BC$
3.  $B \rightarrow b \mid \lambda$
4.  $C \rightarrow D \mid \lambda$
5.  $D \rightarrow d$

Solution:

- $S \rightarrow ABaC \mid AaC \mid ABa \mid Aa \mid BaC \mid aC \mid Ba \mid a$
- $A \rightarrow B \mid C \mid BC$
- $B \rightarrow b$
- $C \rightarrow D$
- $D \rightarrow d$

# Removing unit production

Any production of a CFG of the form

$$A \rightarrow B$$

where  $A, B \in V$  is called a unit production. To remove unit production we use substitution rule.

## Example

$$S \rightarrow Aa \mid B$$

$$B \rightarrow A \mid bb$$

$$A \rightarrow a \mid bc \mid B$$

$$\Rightarrow S \rightarrow Aa$$

$$B \rightarrow bb$$

$$A \rightarrow a \mid bc$$

Then new rule,

$$S \rightarrow Aa \mid bb \mid a \mid bc$$

$$B \rightarrow a \mid bc \mid bb$$

$$A \rightarrow a \mid bc \mid bb$$

Here removal of unit product has made B and the associated production useless.

## Suggested readings

1. An introduction to FORMAL LANGUAGES and AUTOMATA by PETER LINZ.
2. Introduction to Automata Theory, Languages, And Computation by JOHN E. HOPCROFT, RAJEEV MOTWANI, JEFFREY D. ULLMAN
3. Theory of computer science: automata, languages and computation by K.L.P MISHRA

# Thank you

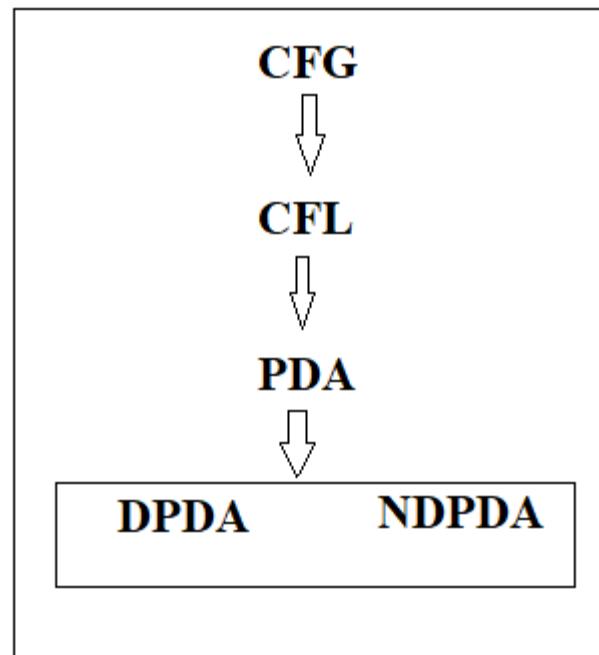
Theory of Computation: CS-202

Context free Language

# Outline

- Pumping Lemma for CFL
- Closure Properties of CFL

# Context free Grammar, Language and PDA



# Pumping Lemma for Context free Language

The pumping lemma gives us a technique to show that certain languages are not context free

- Just like we used the pumping lemma to show certain languages are not regular
- But the pumping lemma for CFL's is a bit more complicated than the pumping lemma for regular languages

# Pumping Lemma for Context free Language

Let  $L$  be an infinite Context free language then there exist some positive Number  $m$  such that any  $w \in L$  with  $|w| \geq m$  can be decomposed as:

$$w = uvxyz$$

with  
&

$$\begin{aligned} |vxy| &\leq m \\ |vy| &\geq 1 \end{aligned}$$

Such that  $uv^i xy^i z \in L , i \geq 0$

# Example

Let  $L$  be the language  $\{ a^n b^n c^n \mid n \geq 1 \}$ . Show that this language is not a CFL.

Let  $m=3$

$w = aabbcc$

Let  $m=3$

$w=a\ a\ a\ b\ b\ bcccc$

u   v   x   y   z

$w=uvxyz$   
 $|vxy| \leq m$   
&       $|vy| \geq 1$

$uv^i xy^i z \notin L$

Here,  $|vy|=2$   
 $|vxy|=3=m$

For  $i=2$

$\Rightarrow aa\ aa\ b\ bb\ bcccc \notin L$

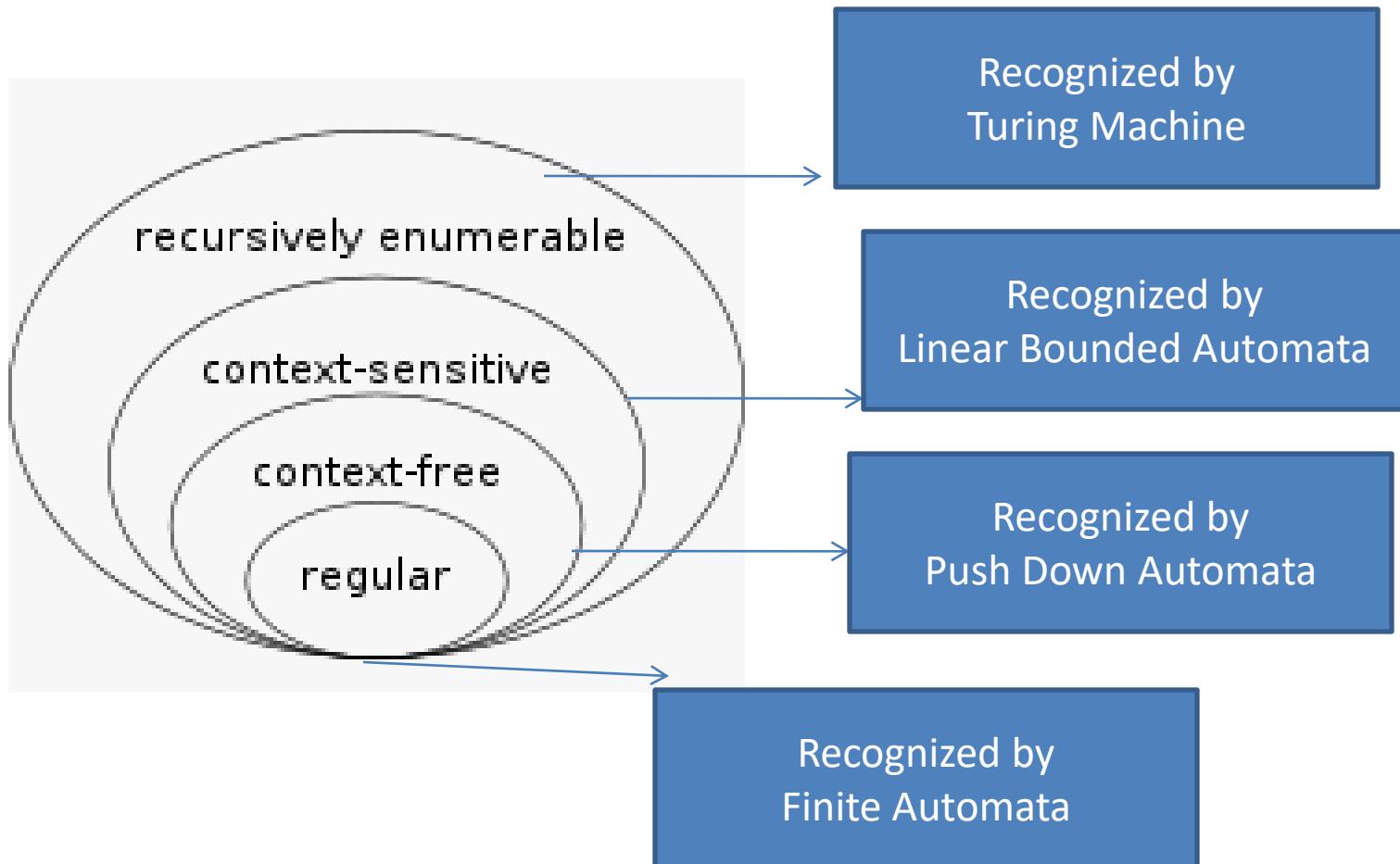
Hence the given language is not Context free

# Closure Properties of Context free Language

The CFL's are closed under union, concatenation, star **closure** , reversal, homomorphism.

CFL's are not closed under intersection and complementation.

# Grammar Hierarchy



# For the Grammar $G=(V, T, S, P)$

Regular Language  
Recognized by  
Finite Automata

Regular Grammar

$$\alpha \rightarrow \beta$$

Where,

$$\alpha \in V$$

$$\beta \in VT^* | T^*$$

$$\text{Or } \beta \in T^* V | T^*$$

Context free Language  
Recognized by  
Push Down Automata

Context free Grammar

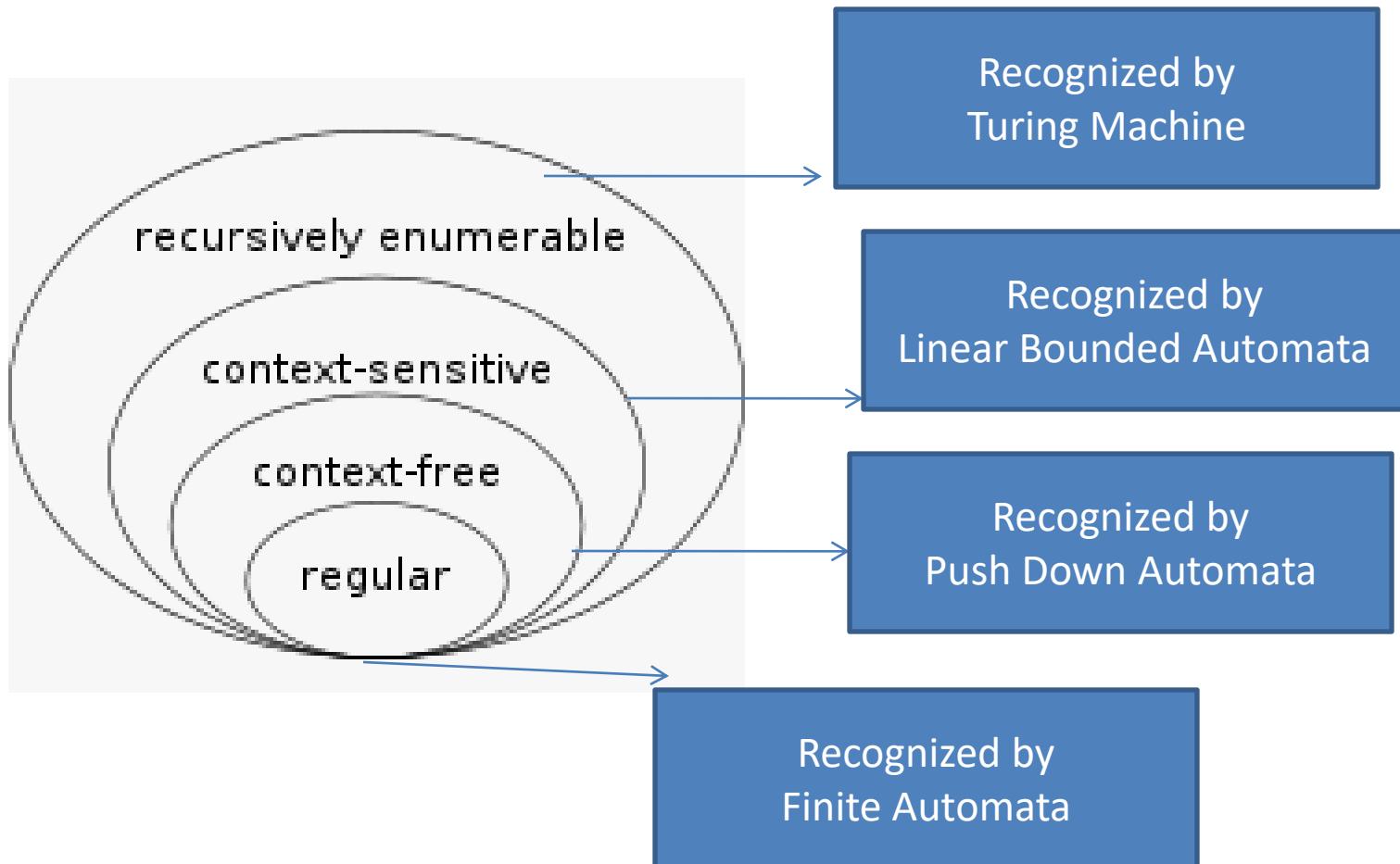
$$\alpha \rightarrow \beta$$

Where,

$$\alpha \in V$$

$$\beta \in (VUT)^*$$

# Grammar Hierarchy



# Context Sensitive Grammar (CSG)

A Grammar  $G=(V, T, S, P)$  is said to be context sensitive if all productions are of the form:

$$\alpha \rightarrow \beta$$

Where,  $\alpha \in (V \cup T)^* V (V \cup T)^*$

$\beta \in (V \cup T)^+$

$|\alpha| \leq |\beta|$

$\Rightarrow$  It does not incorporate  $\epsilon$  productions.

Note: Add  $S \rightarrow \epsilon$  such that  $S$  should not occur twice on the R. H.S of any production.

## Suggested readings

1. An introduction to FORMAL LANGUAGES and AUTOMATA by PETER LINZ.
2. Introduction to Automata Theory, Languages, And Computation by JOHN E. HOPCROFT, RAJEEV MOTWANI, JEFFREY D. ULLMAN
3. Theory of computer science: automata, languages and computation by K.L.P MISHRA

# Thank you

# Theory of Computation: CS-202

## Normal Forms

# Outline

- Two Important Normal Forms
  - Chomsky Normal Form
  - Greibach Normal Form

# Chomsky Normal Form

- A Context free Grammar G is in Chomsky normal form if all the productions are of the form

$A \rightarrow BC$

or  $A \rightarrow a$

Where,  $A, B, C \in V$  and  $a \in T$

# Example

Consider the grammar G with production:

$$S \rightarrow AS \mid a$$

$$A \rightarrow SA \mid b$$

is in Chomsky Normal Form.

# Example

Consider the grammar G with production:

$$S \rightarrow AS \mid AAS$$

$$A \rightarrow SA \mid aS$$

is not in Chomsky Normal Form.

# Example

Convert the grammar  $G=(\{A,B,C\}, \{a,b,c\}, S, P)$  into Chomsky Normal Form.

$P: S \rightarrow AB \ a$

$A \rightarrow aab$

$B \rightarrow Ac$

$\Rightarrow \begin{aligned} S &\rightarrow ABB_a \\ A &\rightarrow B_a B_a B_b \\ B &\rightarrow A B_c \\ B_a &\rightarrow a, \quad B_b \rightarrow b, \quad B_c \rightarrow c \end{aligned}$

$\Rightarrow \begin{aligned} S &\rightarrow AD_1 \\ D_1 &\rightarrow BB_a \\ A &\rightarrow D_2 B_b \\ D_2 &\rightarrow B_a B_a, \quad B \rightarrow A B_c \\ B_a &\rightarrow a, \quad B_b \rightarrow b, \quad B_c \rightarrow c \end{aligned}$

# Greibach Normal Form

A Context free Grammar G is in Greibach Normal Form if all the productions are of the form

$$A \rightarrow a\alpha$$

or  $A \rightarrow a$

Where,  $A \in V$  and  $a \in T$ ,  $\alpha \in V^*$

# Example

Construct a grammar in Greibach normal form equivalent to the grammar

$$S \rightarrow AB, \quad A \rightarrow aA \mid bB/b, \quad B \rightarrow b$$

The given grammar is not in Greibach normal form However, using the substitution rule, we can immediately get the equivalent grammar

$$S \rightarrow aAB/bBB/bB$$

$$A \rightarrow aA/bB/b$$

$$B \rightarrow b$$

*Which is in Greibach normal form*

- For proof please refer to **An introduction to  
FORMAL LANGUAGES and AUTOMATA**  
by PETER LINZ

## Suggested readings

1. An introduction to FORMAL LANGUAGES and AUTOMATA by PETER LINZ.
2. Introduction to Automata Theory, Languages, And Computation by JOHN E. HOPCROFT, RAJEEV MOTWANI, JEFFREY D. ULLMAN
3. Theory of computer science: automata, languages and computation by K.L.P MISHRA

# Thank you

# Theory of Computation: CS-202

## Push Down Automata

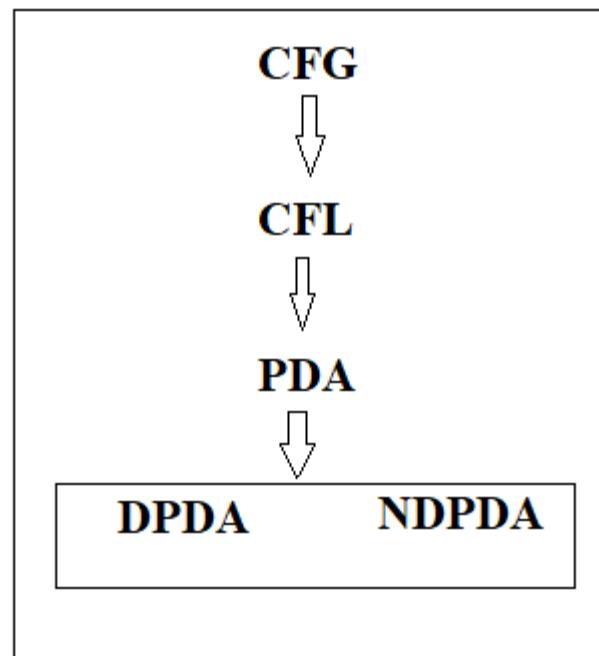
# Outline

- Push Down Automata

- Deterministic Push Down Automata

- Non Deterministic Push Down Automata

# Context free Grammar, Language and PDA



# Formal Definition of a deterministic PDA

A pushdown automaton (PDA) is defined by the seven-tuples:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

$Q$  A finite set of states

---

$\Sigma$  A finite input alphabet

---

---

$\Gamma$  A finite stack alphabet

$q_0$  The initial/starting state,  $q_0$  is in  $Q$

---

---

$z_0$  A starting stack symbol, is in  $\Gamma$

$F$  A set of final/accepting states, which is a subset of  $Q$

---

---

 $z_0$ 

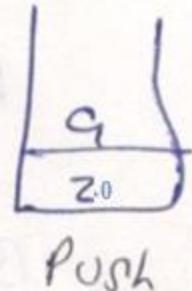
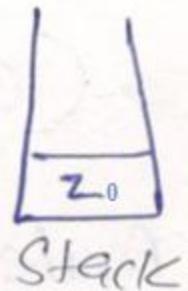
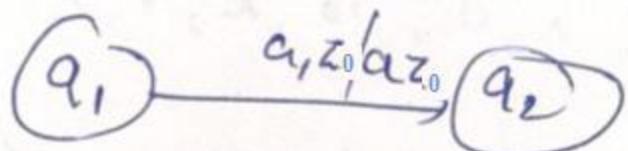
$\delta$  A transition function, where

$$\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow Q \times \Gamma^*$$

# Moves of the PDA: Push, Pop, Skip

Push:

$$\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow Q \times \Gamma^*$$

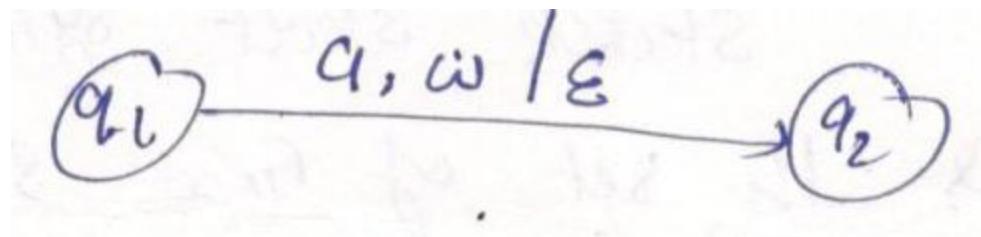


$$\delta(q_1, a, z) = (q_2, az)$$

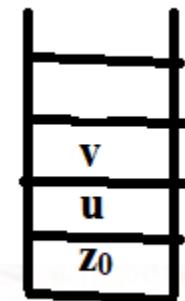
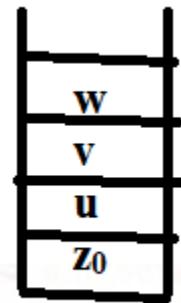
# Moves of the PDA (Cont..)

Pop:

$$\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow Q \times \Gamma^*$$



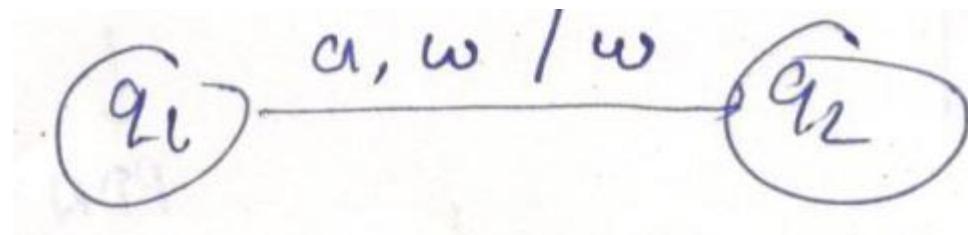
$$\delta(q_1, a, w) = (q_2, \varepsilon)$$



# Moves of the PDA (Cont..)

Skip:

$$\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow Q \times \Gamma^*$$



$$\delta(q_1, a, w) = (q_2, w)$$

# Moves of the PDA (Cont..)

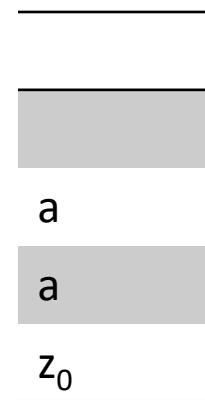
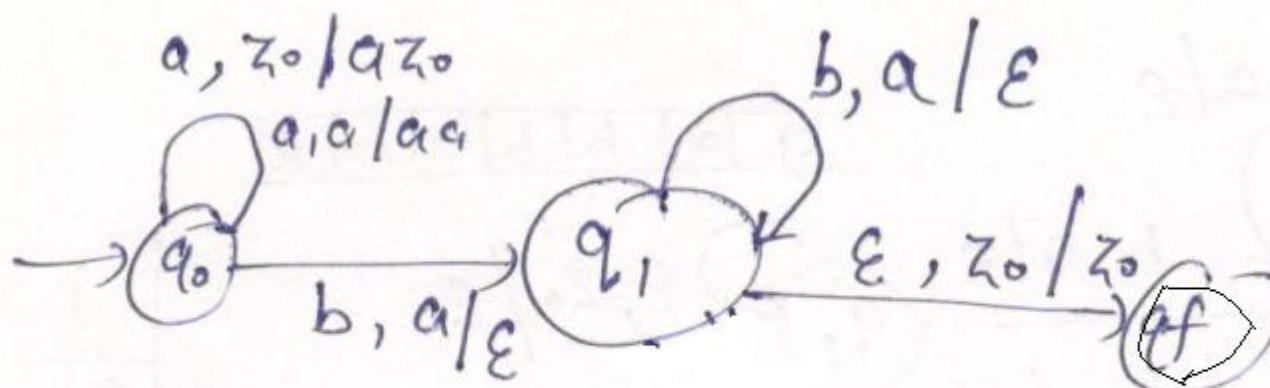
Final:

$$\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow Q \times \Gamma^*$$

$$\delta(q_i, \varepsilon, z_0) = (q_f, z_0)$$

$$\text{Or } \delta(q_i, \varepsilon, z_0) = (q_f, \varepsilon)$$

# Design a PDA for the language

$$L = \{ a^n b^n, n \geq 1 \}$$


# Design a PDA for the language

$$L = \{ a^n b^n, n \geq 1 \}$$

Transition steps for L

$$\delta(q_0, a, z_0) = (q_0, az_0)$$

$$\delta(q_0, a, a) = (q_0, aa)$$

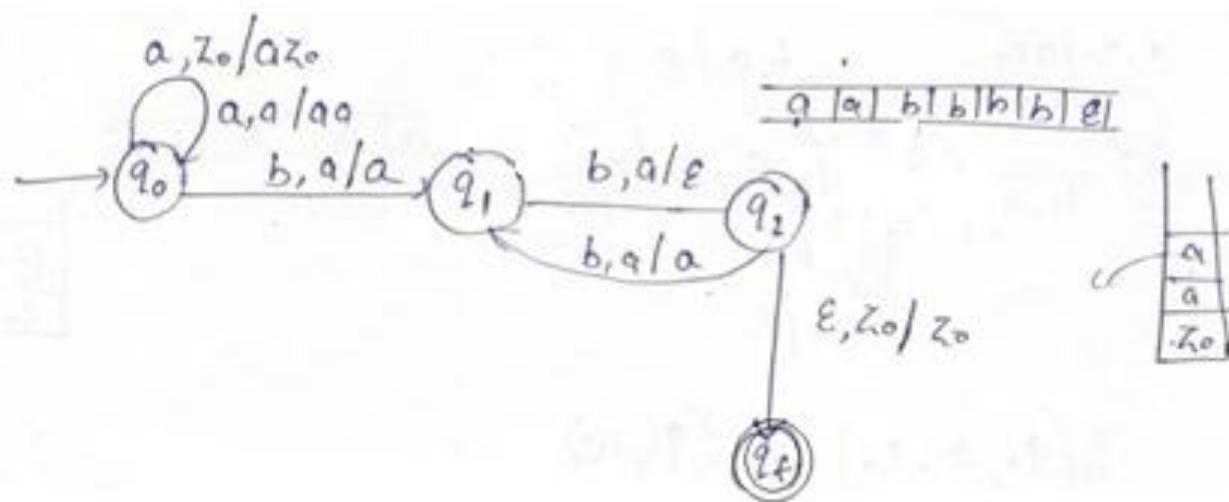
$$\delta(q_0, b, a) = (q_1, \varepsilon)$$

$$\delta(q_1, b, a) = (q_1, \varepsilon)$$

$$\delta(q_1, \varepsilon, z_0) = (q_f, z_0)$$

$$\delta(q_1, \varepsilon, z_0) = (q_f, z_0)$$

# Design a PDA for the language

$$L = \{ a^n b^{2n}, \quad n \geq 0 \}$$


# Design a PDA for the language

$$L = \{ a^n b^{2n}, \quad n \geq 0 \}$$

$$\delta(q_0, a, z_0) = (q_0, az_0)$$

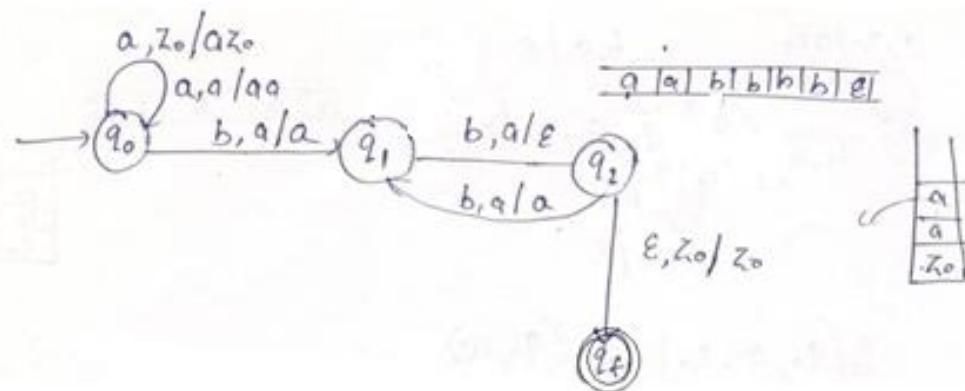
$$\delta(q_0, a, a) = (q_0, aa)$$

$$\delta(q_0, b, a) = (q_1, a)$$

$$\delta(q_1, b, a) = (q_2, \epsilon)$$

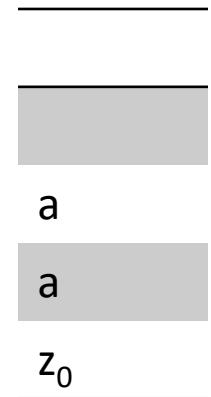
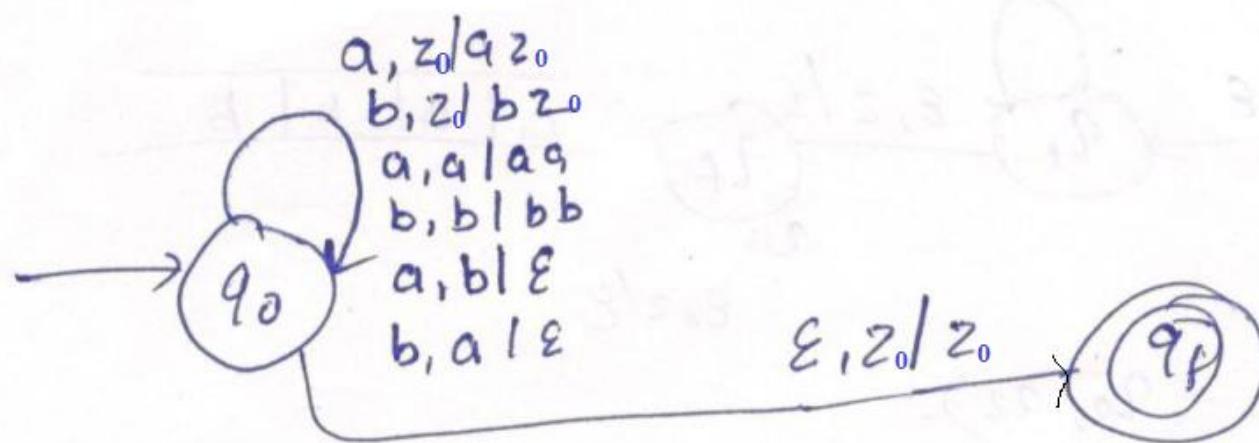
$$\delta(q_2, b, a) = (q_1, a)$$

$$\delta(q_1, \epsilon, z_0) = (q_f, z_0)$$



# Design a PDA for the language

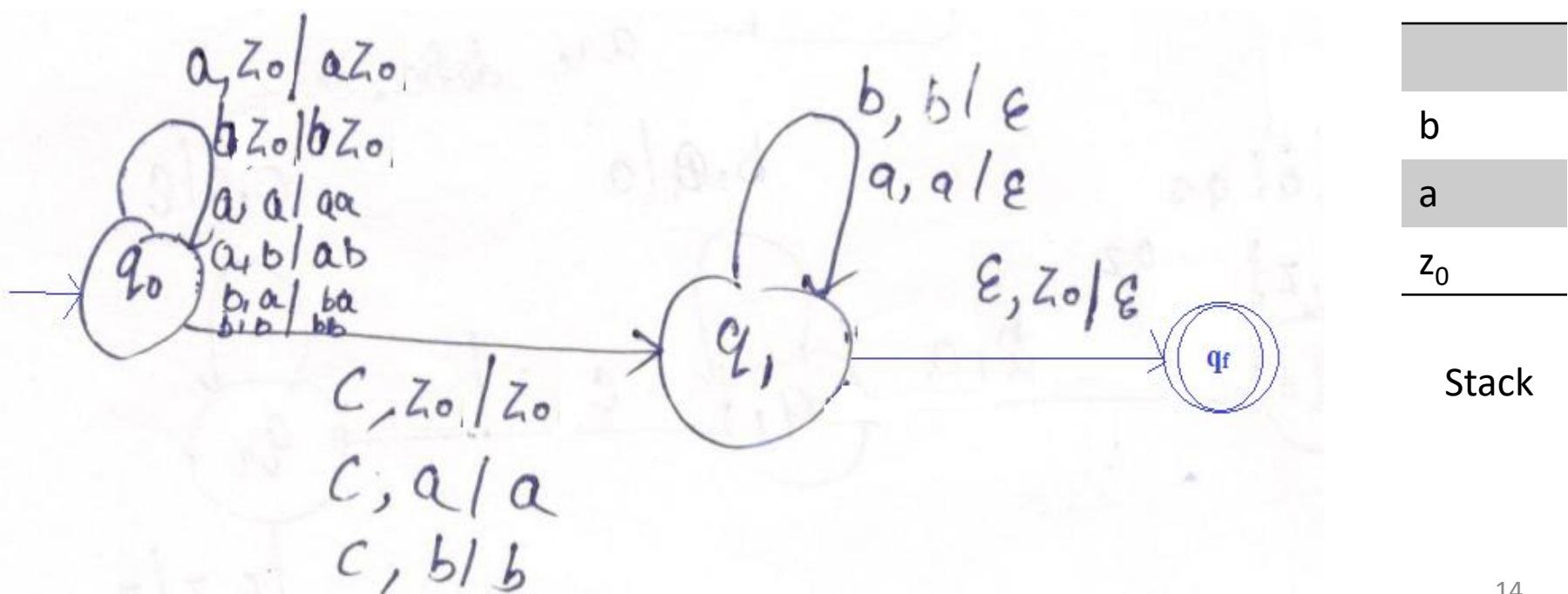
$L = \{ w \in (a, b)^*: n_a(w) = n_b(w) \}$



# Design a PDA for the language

$$L = \{ wczw^R, \quad w \in (a, b)^* \}$$

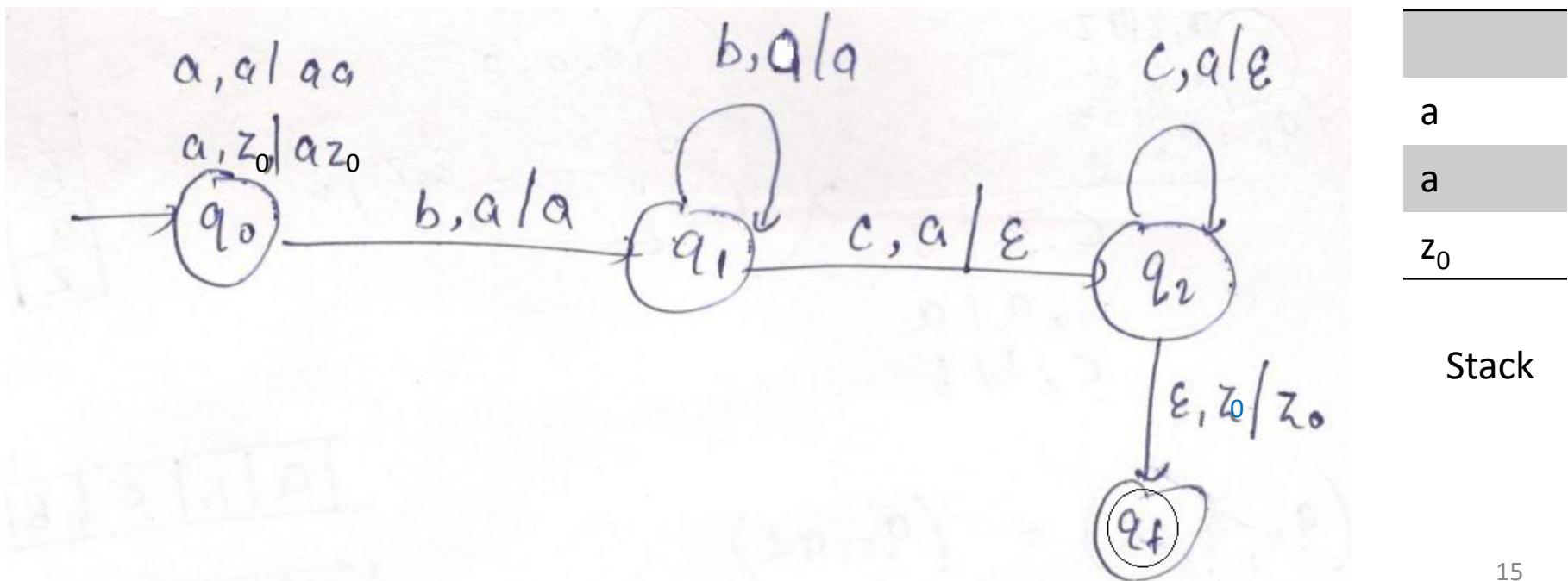
Input tape



# Design a PDA for the language

$$L = \{ a^n b^m c^n, \quad n, m \geq 1 \}$$

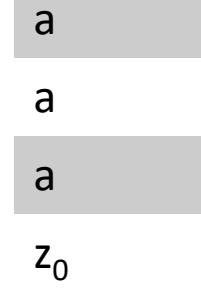
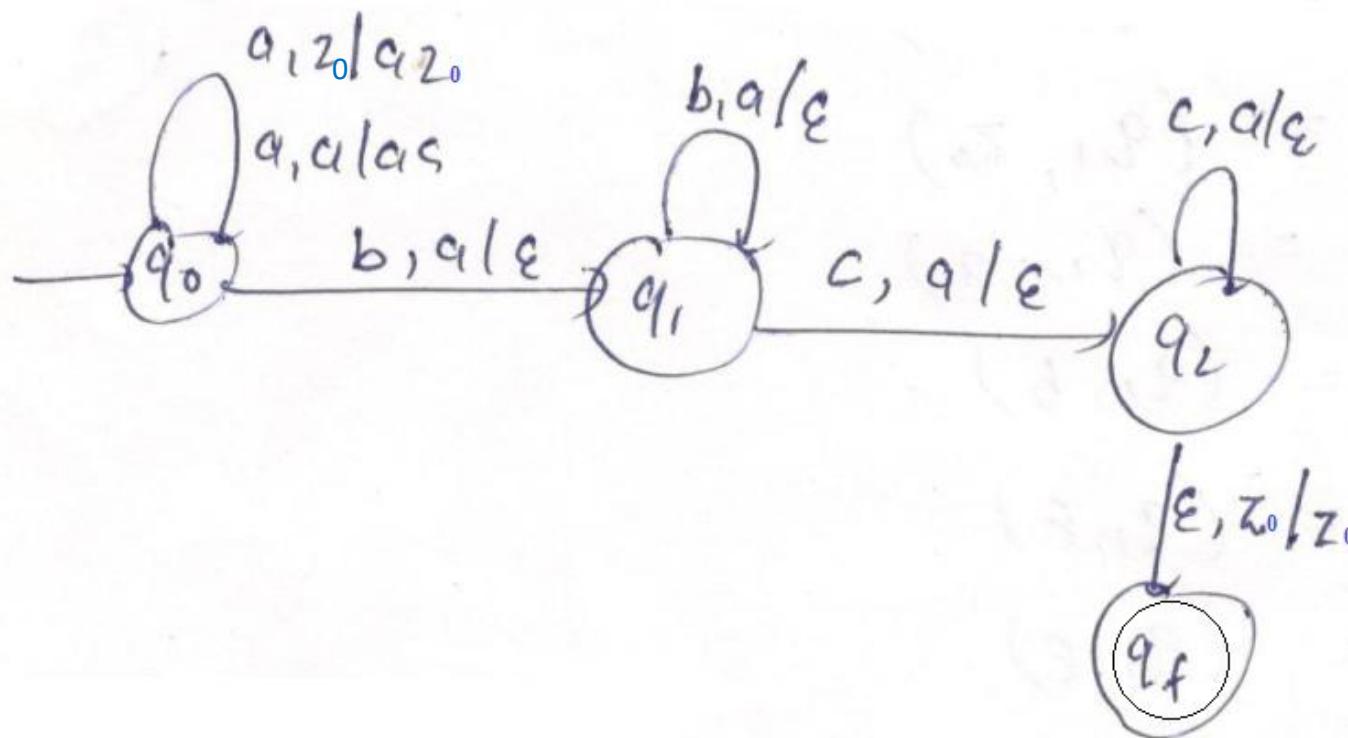
Input tape



# Design a PDA for the language

$$L = \{ a^{m+n} b^m c^n, m, n \geq 1 \}$$

Input tape



Stack

# Practice problem

Design a PDA for the language  $L = \{ a^n b^m c^m d^n \mid m, n \geq 1 \}$



## Suggested readings

1. An introduction to FORMAL LANGUAGES and AUTOMATA by PETER LINZ.
2. Introduction to Automata Theory, Languages, And Computation by JOHN E. HOPCROFT, RAJEEV MOTWANI, JEFFREY D. ULLMAN
3. Theory of computer science: automata, languages and computation by K.L.P MISHRA

# Thank you

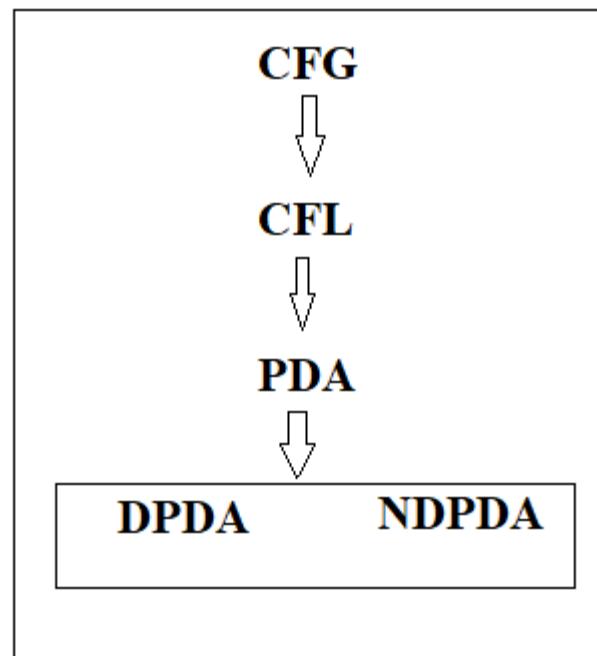
# Theory of Computation: CS-202

## Push Down Automata

# Outline

- Push Down Automata
- Non Deterministic Push Down Automata

# Context free Grammar, Language and PDA



# Formal Definition of a NPDA

A non-deterministic pushdown automaton (NPDA) is defined by the seven-tuples:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

$Q$  A finite set of states

$\Sigma$  A finite input alphabet

$\Gamma$  A finite stack alphabet

$q_0$  The initial/starting state,  $q_0$  is in  $Q$

$z_0$  A starting stack symbol, is in  $\Gamma$

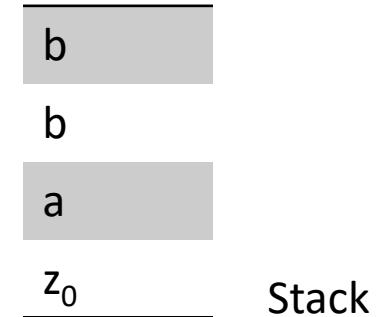
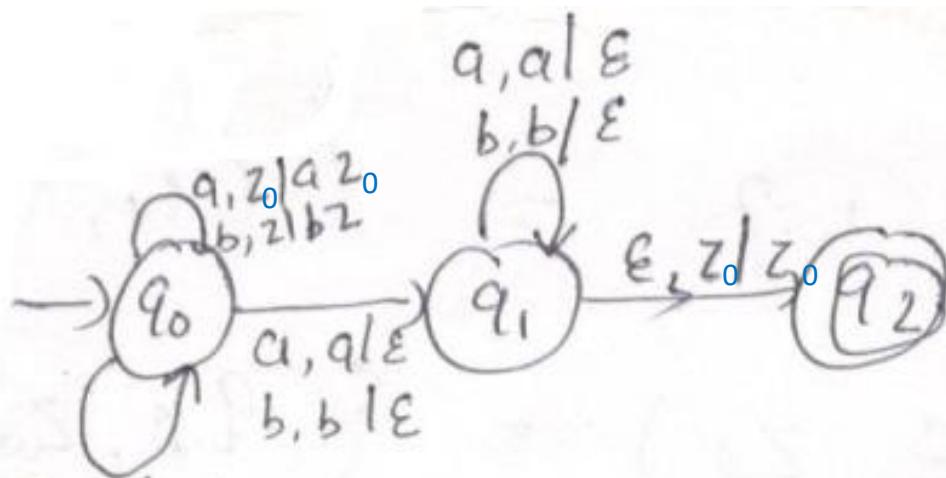
$F$  A set of final/accepting states, which is a subset of  $Q$

$\delta$  A transition function, where

$$\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$$

# Design a NPDA for the language

Input tape       $L = \{ww^R, w \in (a, b)^+\}$



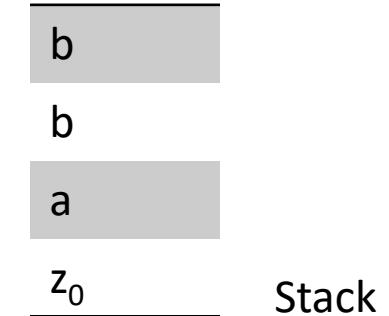
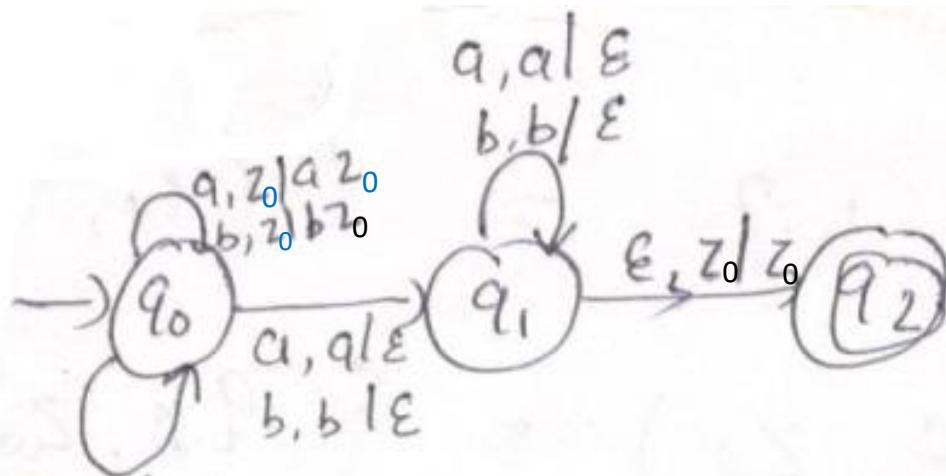
Problem: choosing middle point

Same symbol on stack as well  
as input tape(can assume it centre)  
but not necessarily true.

$a, b | ab$   
 $b, a | ba$   
 $a, a | aa$   
 $b, b | bb$

# Design a NPDA for the language

Input tape       $L = \{ww^R, w \in (a, b)^+\}$



$$\begin{aligned}\delta(q_0, a, a) &= (q_0, aa) \text{ or } (q_1, \epsilon) \\ \delta(q_0, b, b) &= (q_0, bb) \text{ or } (q_1, \epsilon)\end{aligned}$$

These moves make it NPDA

# Power of DPDA and NPDA

Non Deterministic Pushdown Automata (NDPDA) is more powerful than Deterministic Pushdown Automata (DPDA).

# Equivalence between CFG and PDA

**CFG and PDA** are **equivalent** in power:  
a **CFG** generates a context-free language and  
a **PDA** recognizes a context-free language.

A language is context-free iff some pushdown automaton recognizes it.

# Practice Problem

Design a NPDA for the language  
 $L = \{ w b w^R, \quad w \in (a, b)^+ \}$

## Suggested readings

1. An introduction to FORMAL LANGUAGES and AUTOMATA by PETER LINZ.
2. Introduction to Automata Theory, Languages, And Computation by JOHN E. HOPCROFT, RAJEEV MOTWANI, JEFFREY D. ULLMAN
3. Theory of computer science: automata, languages and computation by K.L.P MISHRA

# Thank you

# Theory of Computation: CS-202

## Turing Machine

# Outline

- Standard Turing Machine
- Examples

# Standard Turing Machine

A Turing Machine is an automaton whose temporary storage is a tape, which is divided into cells, each of which is capable of holding one symbol

# Formal Definition of a Standard Turing machine

A Turing machine (TM) is defined by the seven-tuples:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$$

$Q$  A finite set of internal states

$\Sigma$  A finite set of input alphabet

$\Gamma$  A finite set of symbols called tape alphabet

$q_0$  The initial/startling state,  $q_0$  is in  $Q$

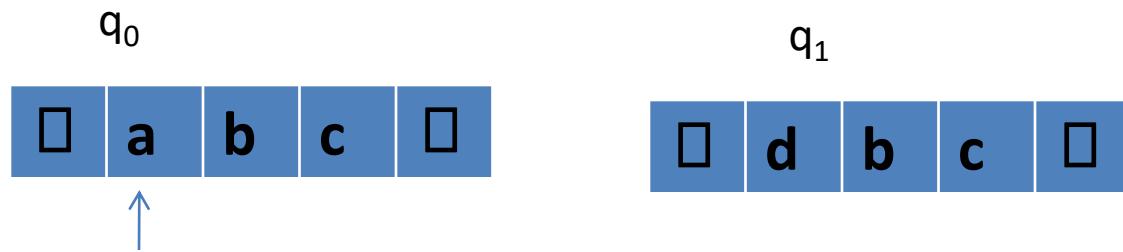
$\square$  A special symbol called the blank symbol, is in  $\Gamma$

$F$  A set of final/accepting states, which is a subset of  $Q$

$\delta$  A transition function, where

$$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L/R\}$$

# Moves of the Turing Machine



$$\delta(q_0, a) = (q_1, d, R)$$

# Consider a Turing machine defined by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$$

$$Q = \{q_0, q_1\}$$

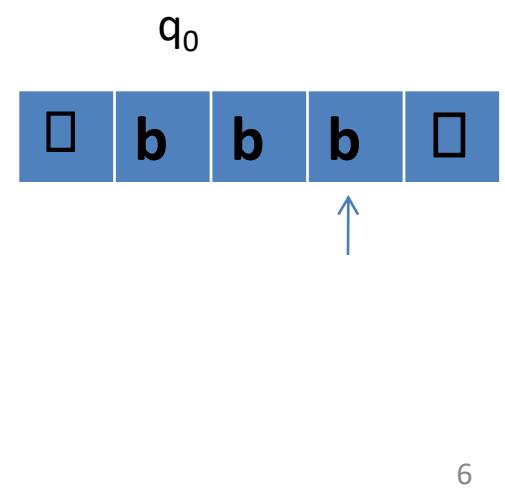
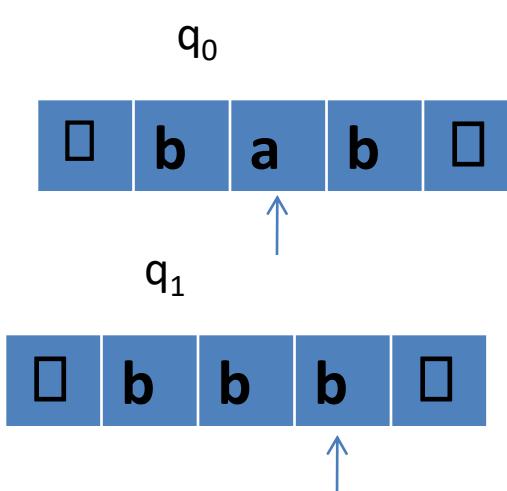
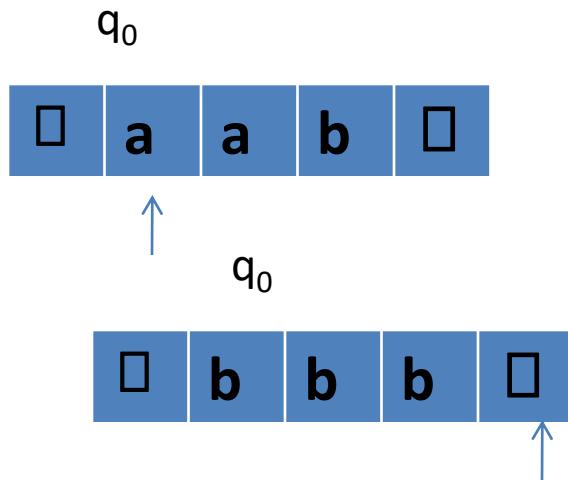
$$\Sigma = \{a, b\}$$

$$\Gamma = \{a, b, \square\}$$

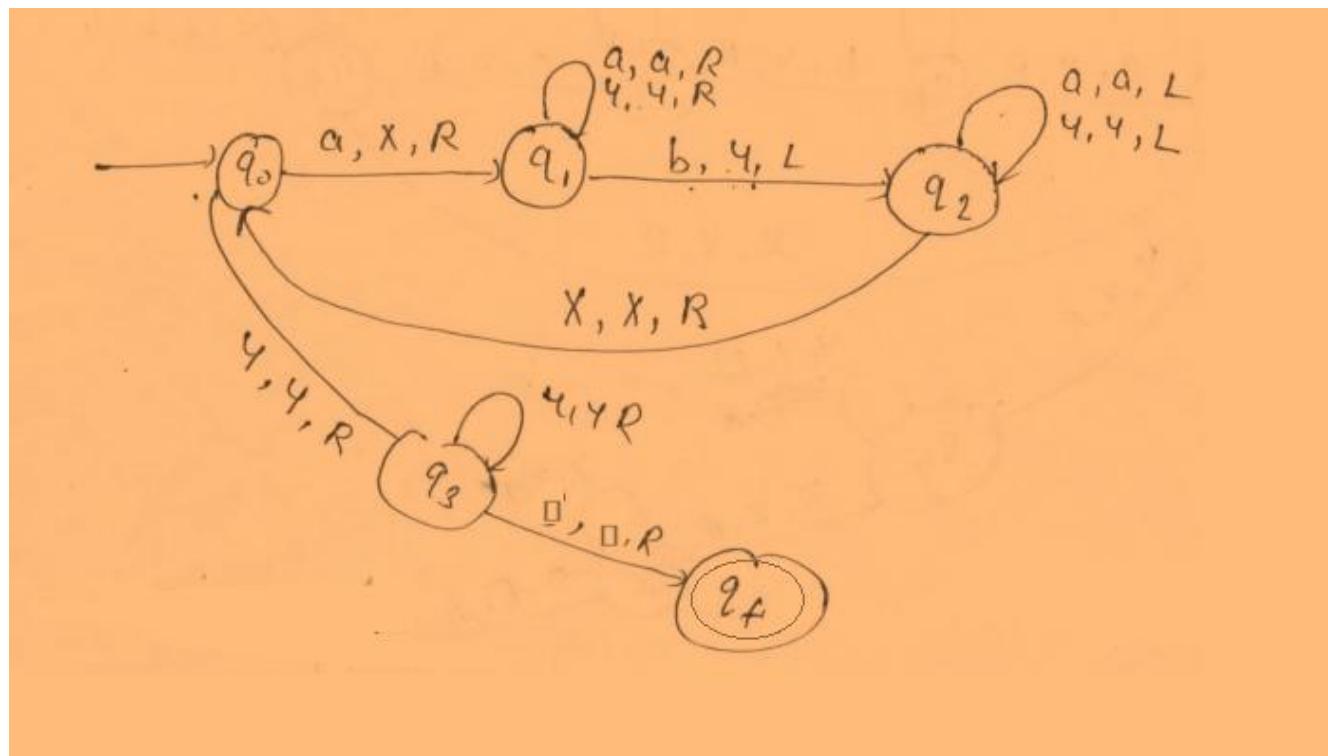
$$F = \{q_1\}$$

$$\& \delta(q_0, a) = (q_0, b, R),$$

$$\delta(q_0, b) = (q_0, b, R), \quad \delta(q_0, \square) = (q_1, \square, L)$$

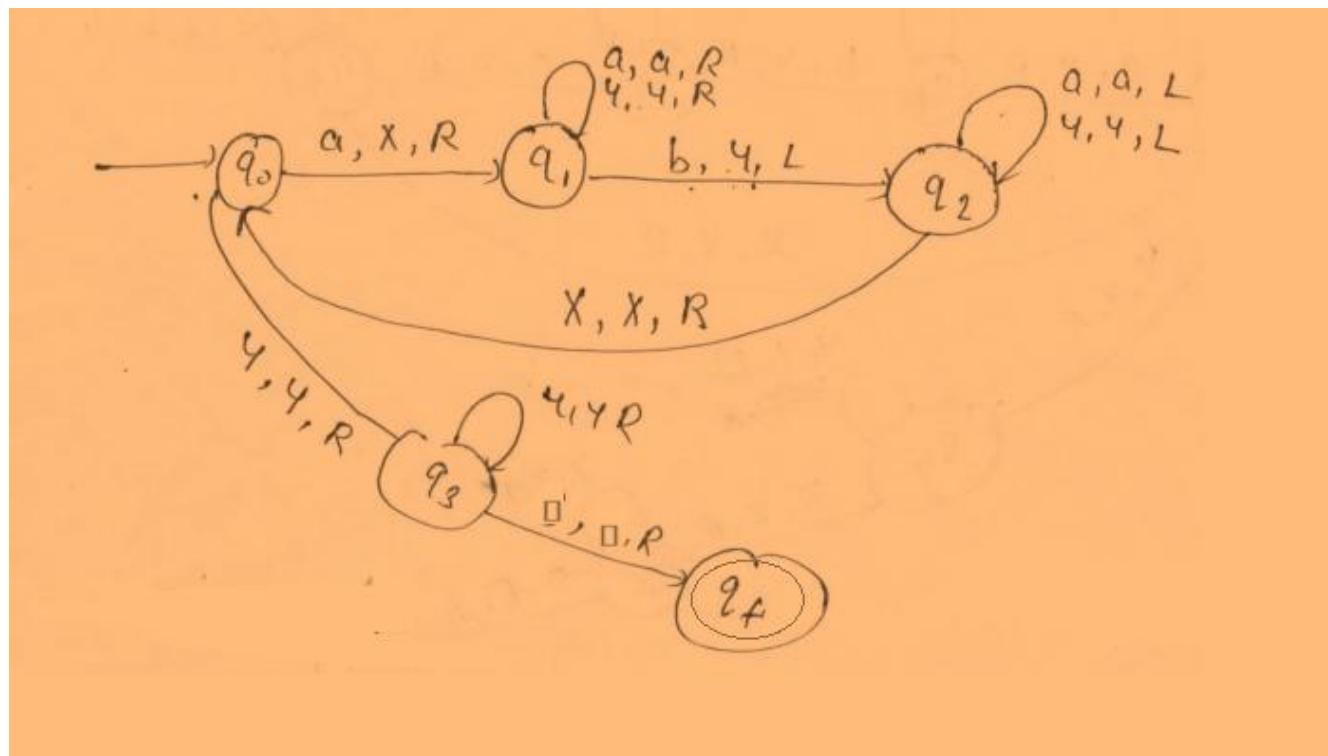


# Design a Turing Machine for the language $L=\{ a^n b^n, \quad n \geq 1 \}$



# Transition Table

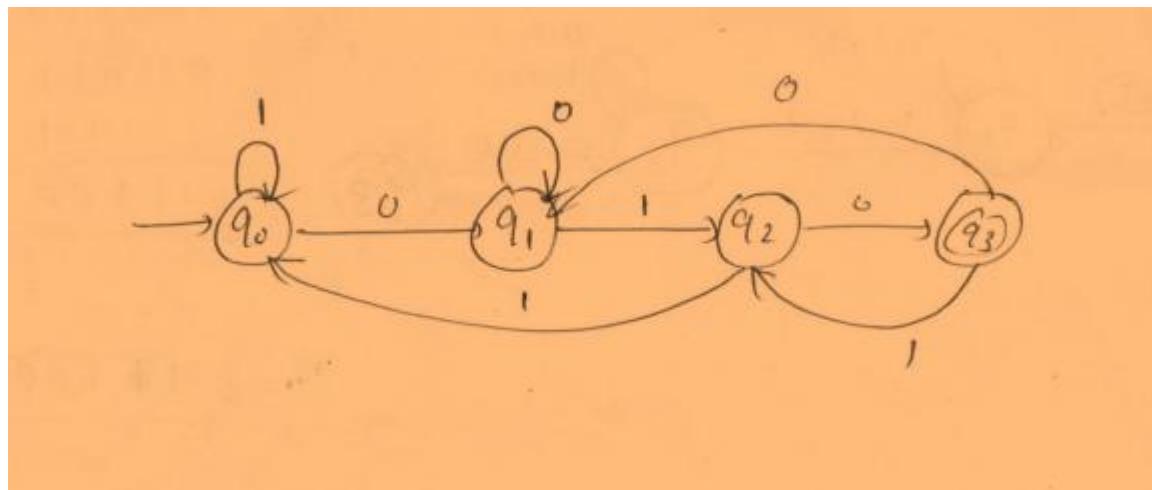
	a	b	X	Y	$\square$
$q_0$	$(q_1, X, R)$	-	-	$(q_3, Y, R)$	-
$q_1$	$(q_1, a, R)$	$(q_2, Y, L)$	-	$(q_1, Y, R)$	-
$q_2$	$(q_2, a, L)$	-	$(q_0, X, R)$	$(q_2, Y, L)$	-
$q_3$	-	-	-	$(q_3, Y, R)$	$(q_f, \square, R)$
$q_f$	-	-	-	-	-



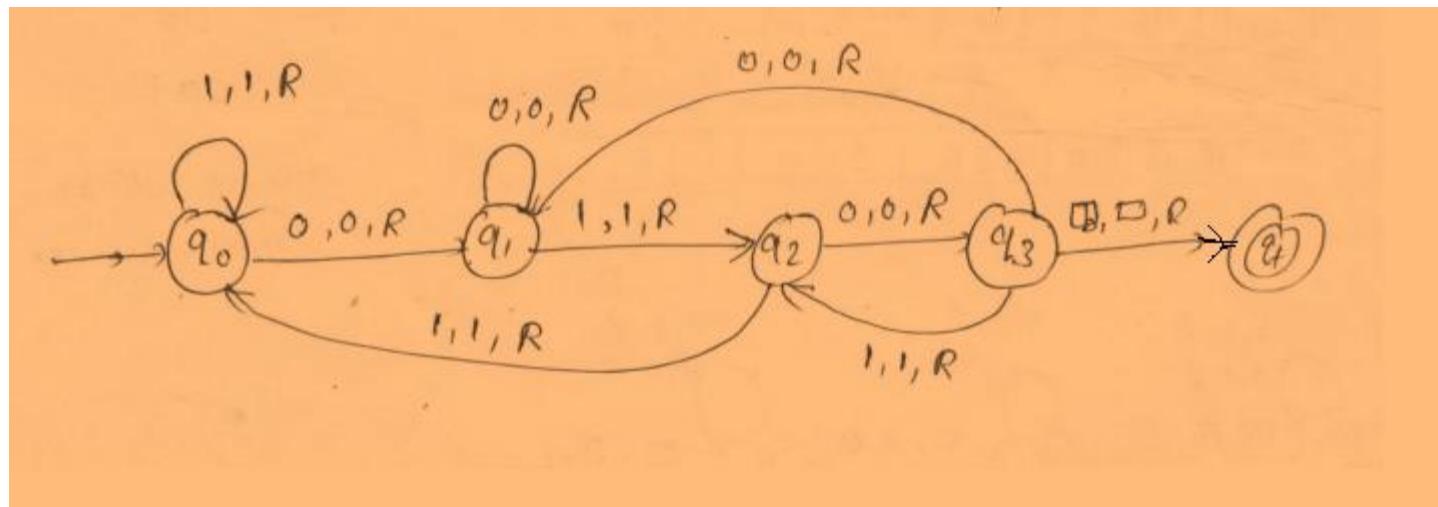
# Design a Turing Machine that accepts a set of string ending with 010



DFA



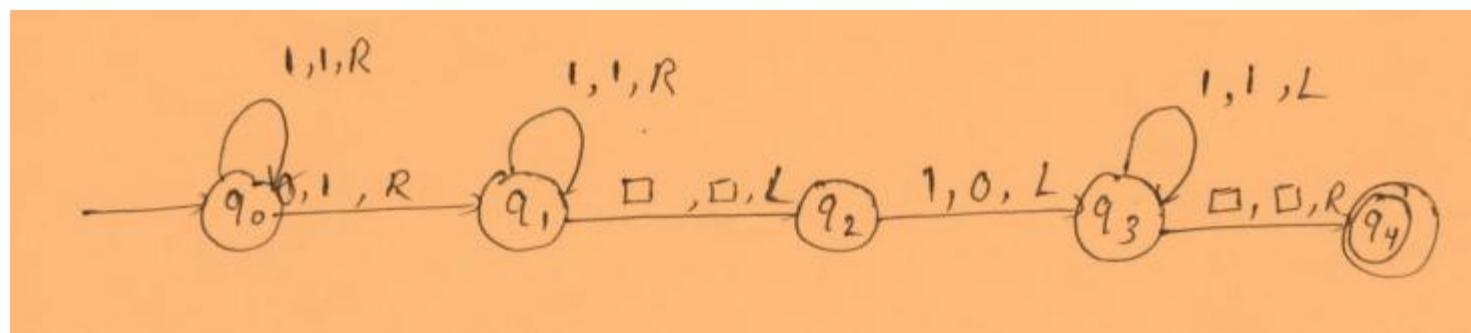
# Design a Turing Machine that accepts a set of string ending with 010



# Design a Turing Machine to add two positive integers

1 | 1 | 1 | 0 | 1 | 1 | □

X=3  
Y=2



# Design a Turing Machine to add two positive integers

1	1	1	0	1	1	□
---	---	---	---	---	---	---

$$\delta(q_0, 1) = (q_0, 1, R)$$

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

$$(q_1, 1) = (q_1, 1, R)$$

$$(q_1, \square) = (q_2, \square, L)$$

$$(q_2, 1) = (q_3, 0, L)$$

$$(q_3, 1) = (q_3, 1, L)$$

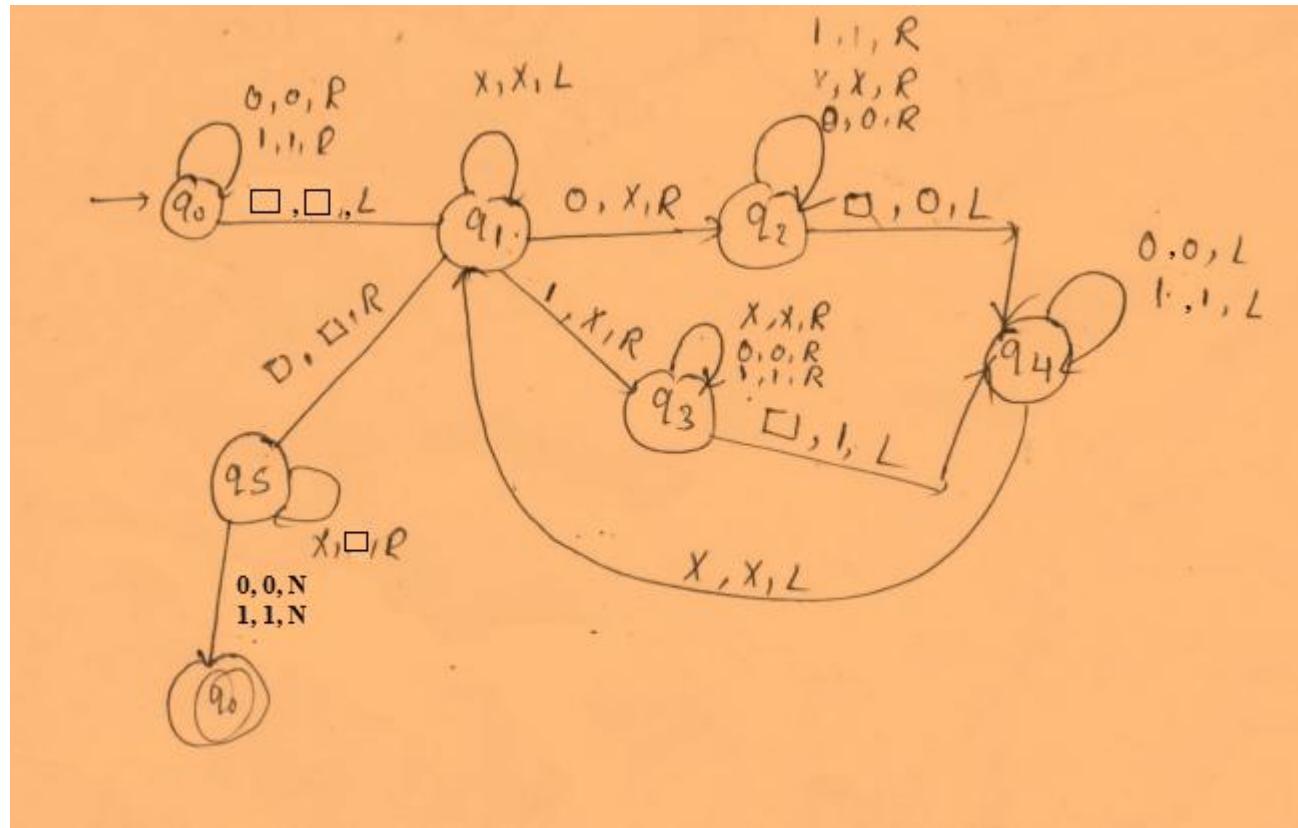
$$(q_3, \square) = (q_4, \square, R)$$

$$q_0 \mid \mid \mid 0 \mid \mid \vdash q_0 \mid \mid 0 \mid \mid \vdash \mid \mid q_0 \mid 0 \mid \mid \vdash \mid \mid \mid q_0 \mid 0 \mid \mid \vdash \mid \mid \mid q_0 \mid 0 \mid \mid \vdash$$

$$\vdash \mid \mid \mid \mid \mid q_1 \mid \vdash \mid \mid \mid \mid \mid q_1 \mid \vdash \mid \mid \mid \mid \mid q_1 \mid \vdash \mid \mid \mid \mid \mid q_1 \mid \vdash$$

$$\vdash \mid \mid \mid \mid \mid q_2 \mid \vdash \mid \mid \mid \mid \mid q_2 \mid \vdash \mid \mid \mid \mid \mid q_2 \mid \vdash \mid \mid \mid \mid \mid q_2 \mid \vdash$$

# Design a Turing Machine that computes string reversal



## Suggested readings

1. An introduction to FORMAL LANGUAGES and AUTOMATA by PETER LINZ.
2. Introduction to Automata Theory, Languages, And Computation by JOHN E. HOPCROFT, RAJEEV MOTWANI, JEFFREY D. ULLMAN
3. Theory of computer science: automata, languages and computation by K.L.P MISHRA

# Thank you

# Theory of Computation: CS-202

## Turing Machine

# Outline

- Variants of Turing machine
  - Multi-tape Turing Machine
  - Multi-Track Turing Machine
  - Non-Deterministic Turing Machine
  - Semi-infinite Tape Turing Machine

# Variants of Turing Machine

- ❑ Non-Deterministic Turing Machine
- ❑ Multi-tape Turing Machine
- ❑ Multi-Track Turing Machine
- ❑ Semi-infinite Tape Turing Machine

# Non-Deterministic Turing Machine

A Turing machine (TM) is defined by the seven-tuples:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$$

$Q$  A finite set of internal states

$\Sigma$  A finite set of input alphabet

$\Gamma$  A finite set of symbols called tape alphabet

$q_0$  The initial/starting state,  $q_0$  is in  $Q$

$\square$  A special symbol called the blank symbol, is in  $\Gamma$

$F$  A set of final/accepting states, which is a subset of  $Q$

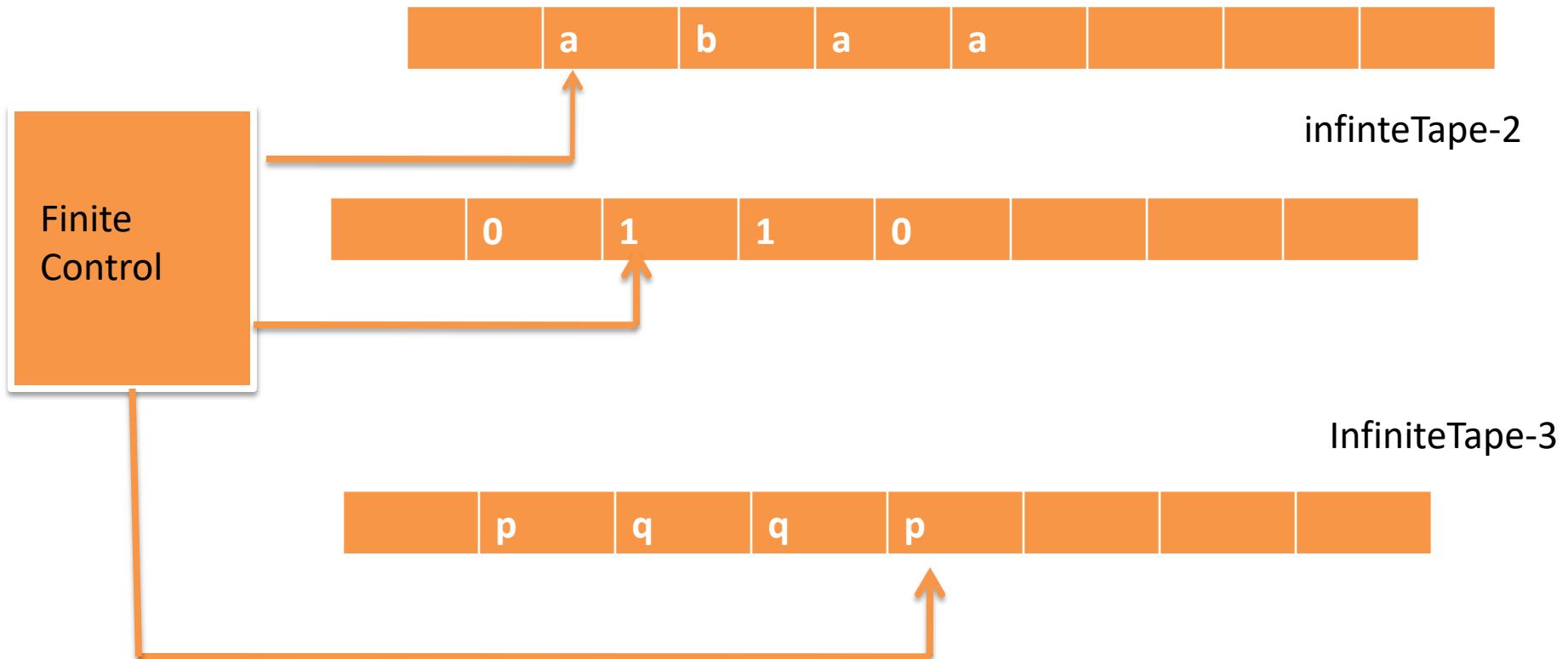
$\delta$  A transition function, where

$$\delta: Q \times \Gamma \rightarrow 2^Q \times \Gamma \times \{L/R\}$$

# Multi-tape Turing Machine

Each tape has its own Read/ write head

infinteTape-1



$$\delta(q_0, a1p) = (q_1, a0q, RRL)$$

# Multi-track Turing Machine



Single tape is divided into multiple tracks

# Semi infinite Tape Turing Machine



One end is infinite

# Equivalence of Turing Machine

Every Multi-Tape Turing Machine has an equivalent Single Tape Turing Machine

## Suggested readings

1. An introduction to FORMAL LANGUAGES and AUTOMATA by PETER LINZ.
2. Introduction to Automata Theory, Languages, And Computation by JOHN E. HOPCROFT, RAJEEV MOTWANI, JEFFREY D. ULLMAN
3. Theory of computer science: automata, languages and computation by K.L.P MISHRA

# Thank you