

A portrait of Dr. N. S. Ramesh, a man with dark hair and a mustache, wearing a dark pinstriped suit, white shirt, and patterned tie. He is looking slightly to the right of the camera.

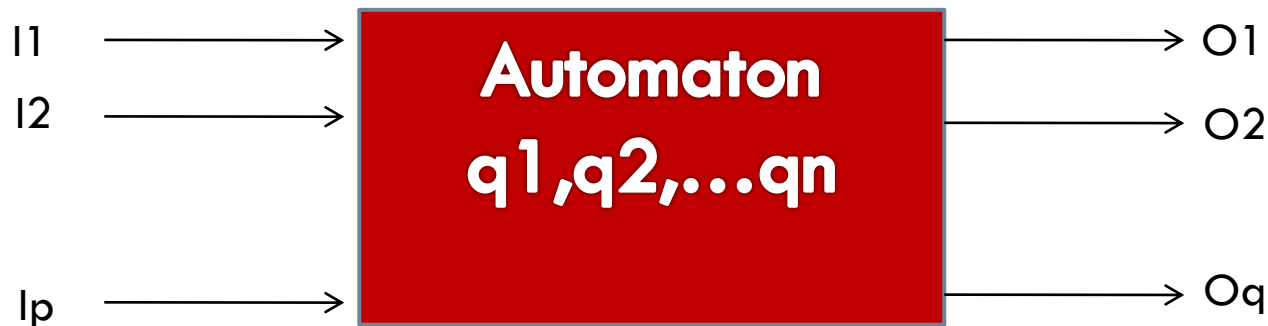
# What is Automata?

- The term "Automata" is derived from the Greek word "αὐτόματα" which means "self-acting".
- It is the plural of automaton, it means "something that works automatically"
- **A system where energy, materials and information, are transformed for performing some specific task, without direct participation of man.**
- **Example:** Automatic photo printing machine, Packing machine, etc.

# Model of Discrete Automaton



- In Computer Science, **Automaton** = an abstract computing device which process discrete information.



**Input:**  $I1, I2, \dots, Ip$

**Output:**  $O1, O2, \dots, Oq$

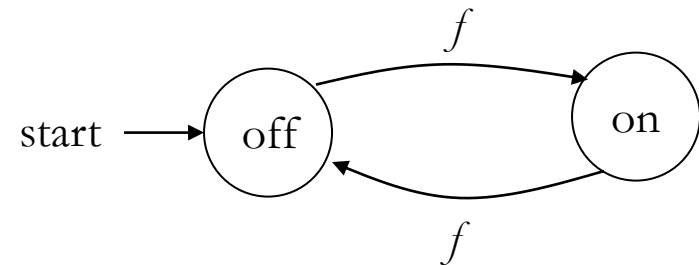
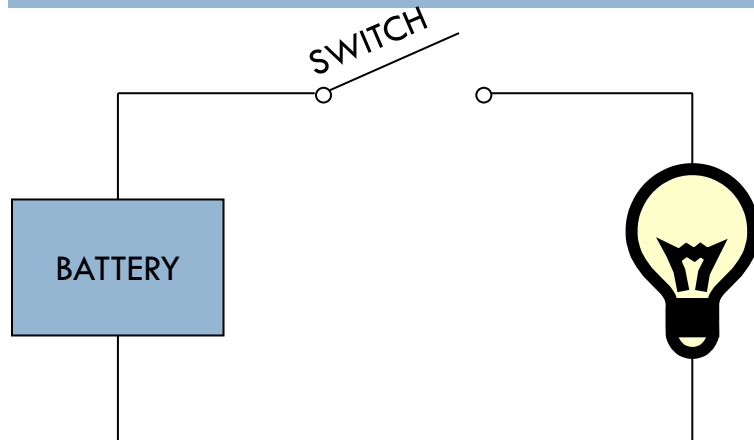
**States:**  $q1, q2, \dots, qn$

# Why do we need abstract models?



- ▮ Abstract model is **free from “Programming Language”**
- ▮ It's **easy to manipulate** these theoretical machines **mathematically** to prove things about their capabilities.

# A simple “computer”



**input:** switch

**output:** light bulb

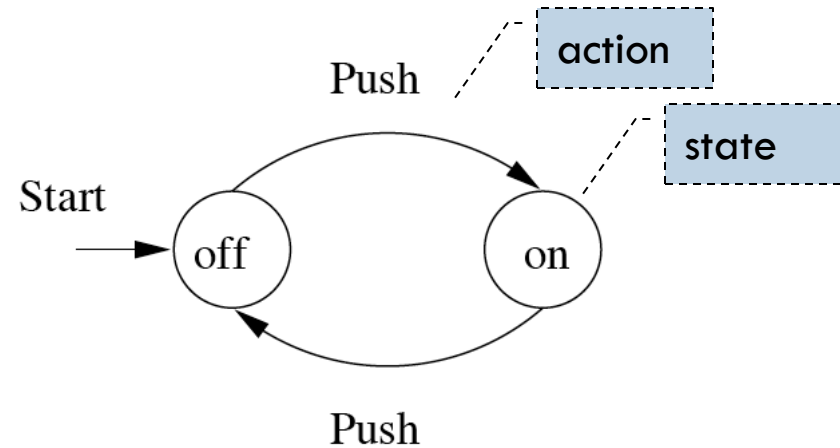
**actions:**  $f$  for “flip switch”

**states:** on, off

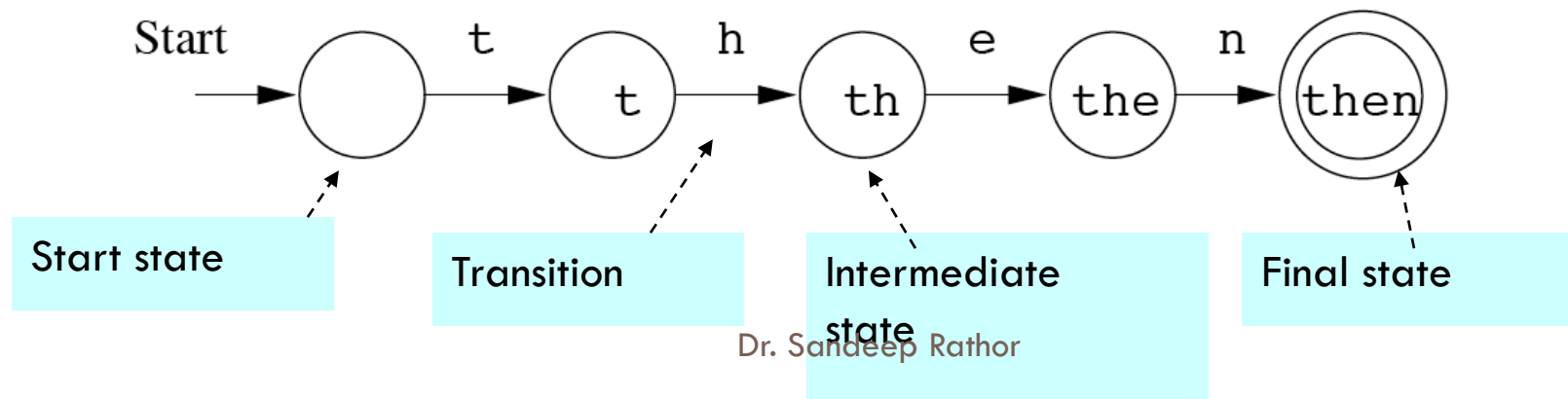
bulb is on if and only if  
there was an **odd** number  
of flips

# Finite Automata : Examples

## □ On/Off switch



## □ Modeling recognition of the word “then”

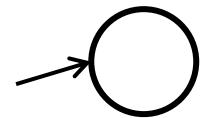


# Transition Diagram

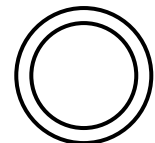


- Directed graph consists of set of vertices and edges where vertices represent “states” and edges represent “input/output”

- Circle with an arrow is called initial state



- Two concentric circle represents the final state



# Alphabets



✧ A finite non empty set of symbols.

Symbol :  $\Sigma$

1. English lang. small letter  $\Sigma = \{a, b, c \dots z\}$
2. Binary number  $\Sigma = \{0, 1\}$
3. Decimal number  $\Sigma = \{0, 1, 2, \dots, 9\}$
4. Alphanumeric :  $\Sigma = \{A-Z, a-z, 0-9\}$



# Strings

- ⌘ A word or a string is a finite sequence of symbols taken from  $\Sigma$ .
- ⌘ Empty string :  $\epsilon$
- ⌘ Length of string  $w$  is denoted by  $|w|$  is number of non empty characters in the string.

Ex.  $x = 01000 \quad |x| = 5$

$x = 01\epsilon 01\epsilon 00\epsilon \quad |x| = ?$

- ⌘  $xy =$  concatenation of two string.

# Strings



A **string** over alphabet  $\Sigma$  is a finite sequence of symbols in  $\Sigma$ .

- The **empty string** will be denoted by  $\varepsilon$
- Examples

abfbz is a string over  $\Sigma_1 = \{a, b, c, d, \dots, z\}$

9021 is a string over  $\Sigma_2 = \{0, 1, \dots, 9\}$

ab#bc is a string over  $\Sigma_3 = \{a, b, \dots, z, \#\}$

))()( is a string over  $\Sigma_4 = \{ (, ) \}$

# Languages

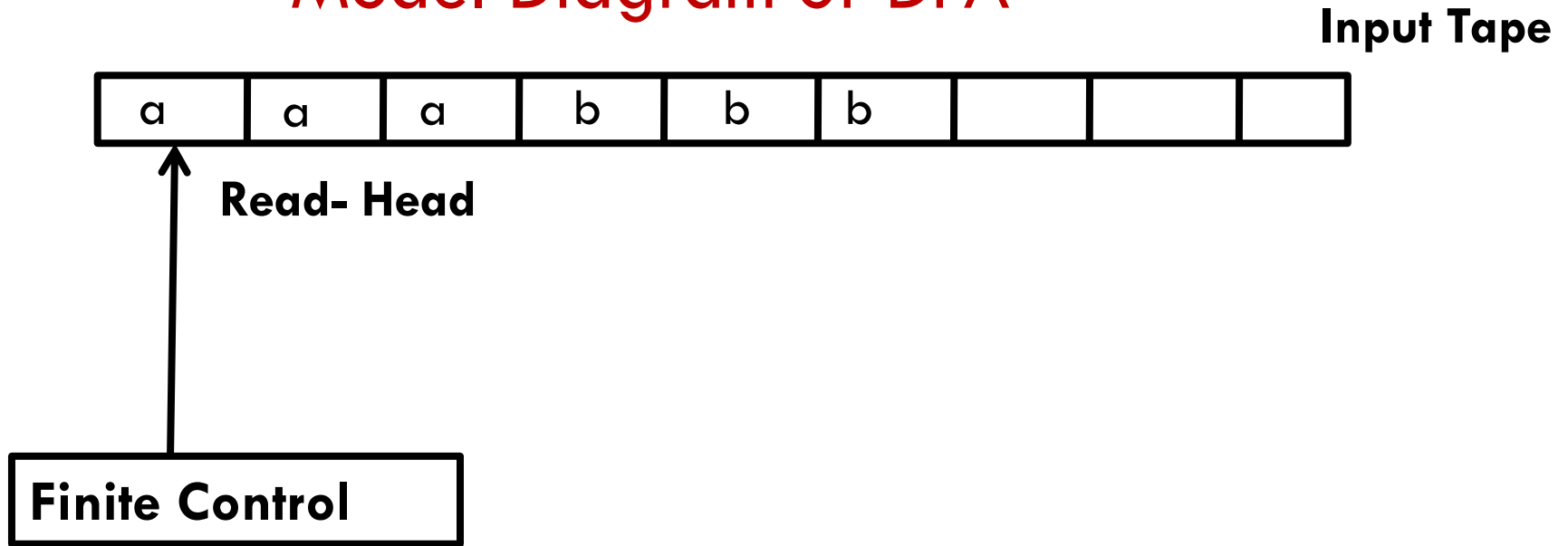


- ⌘ L is said to be a language over alphabet  $\Sigma$  only if  $L \subseteq \Sigma^*$ .
- ⌘  $\Sigma^*$  Set of all string over the  $\Sigma$

# Deterministic Finite Automata (DFA)



## Model Diagram of DFA



\*Warren McCulloch and Walter Pitts were among the first researchers to introduce a concept similar to finite automata in 1943.

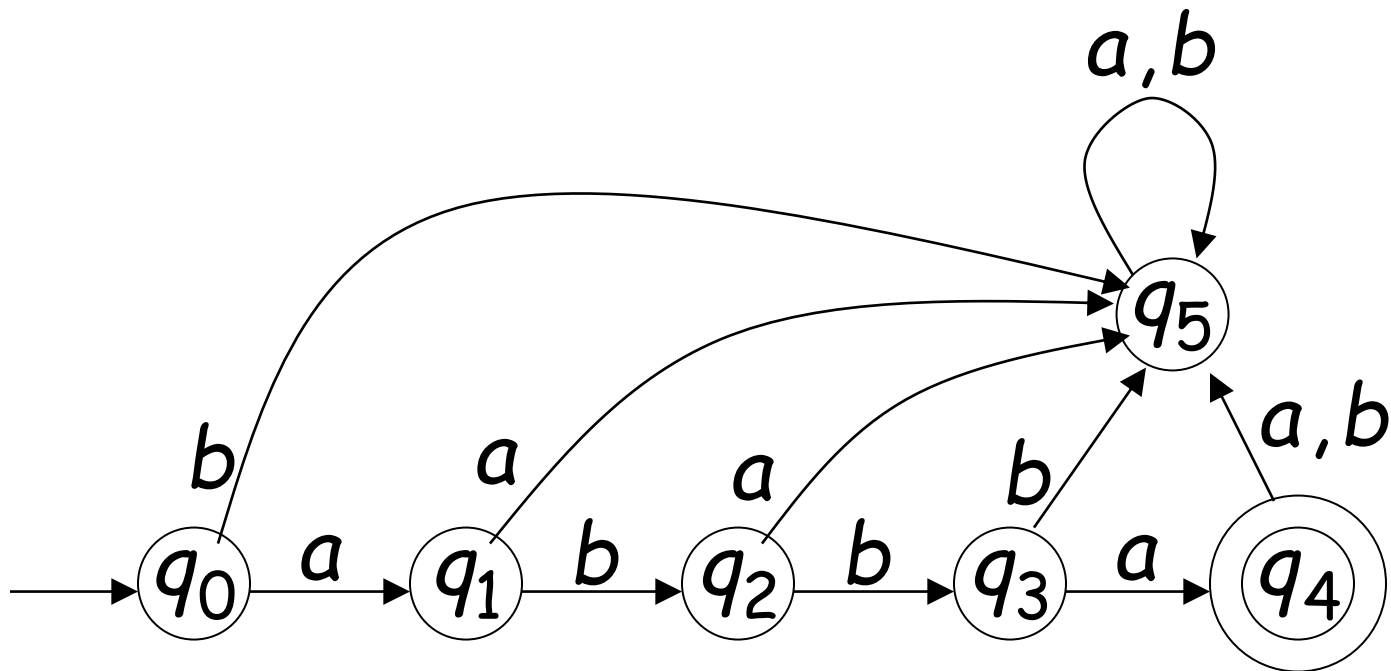
- A **deterministic finite automaton** (DFA) is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$  where
  - $Q$  is a finite set of **states**
  - $\Sigma$  is an input **alphabet**
  - $\delta: Q \times \Sigma \rightarrow Q$  is a **transition function**
  - $q_0 \in Q$  is the **initial state**
  - $F \subseteq Q$  is a set of **accepting states** (or **final states**).

*Note: Deterministic refers to the uniqueness of the computation run.*

# Input Alphabet

 $\Sigma$ 

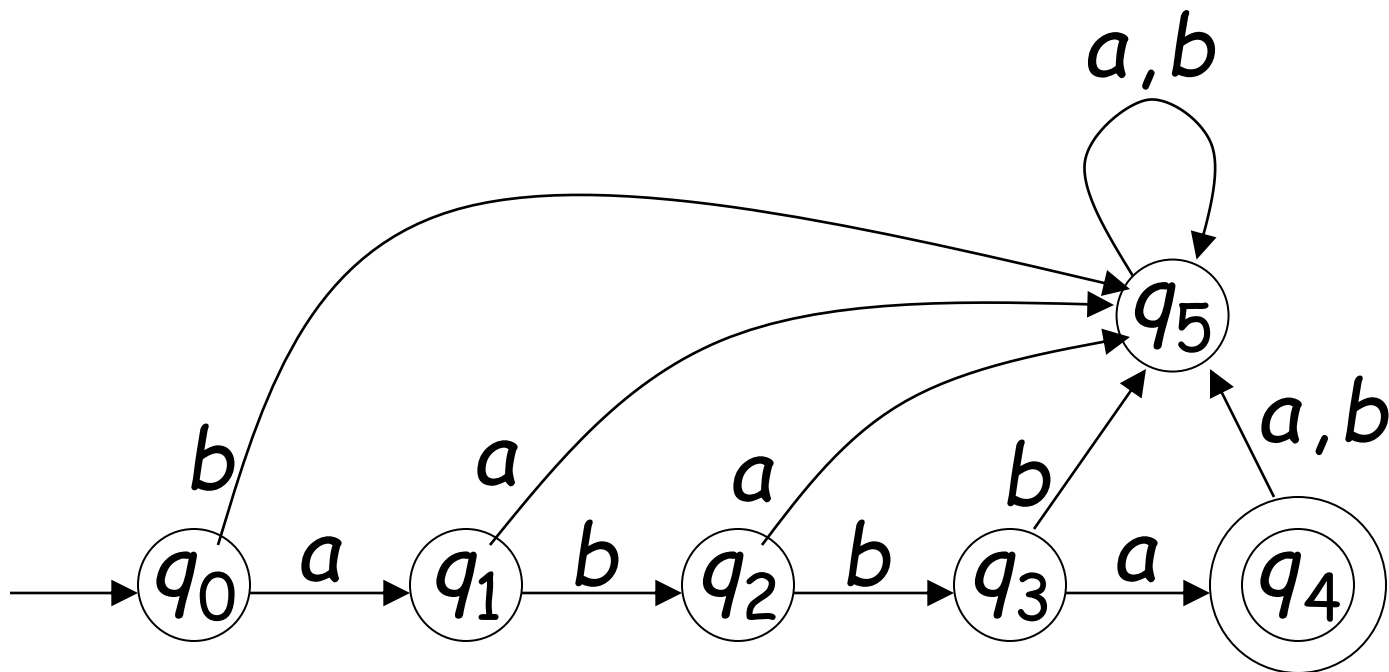
□  $\Sigma = \{a, b\}$



# Set of States

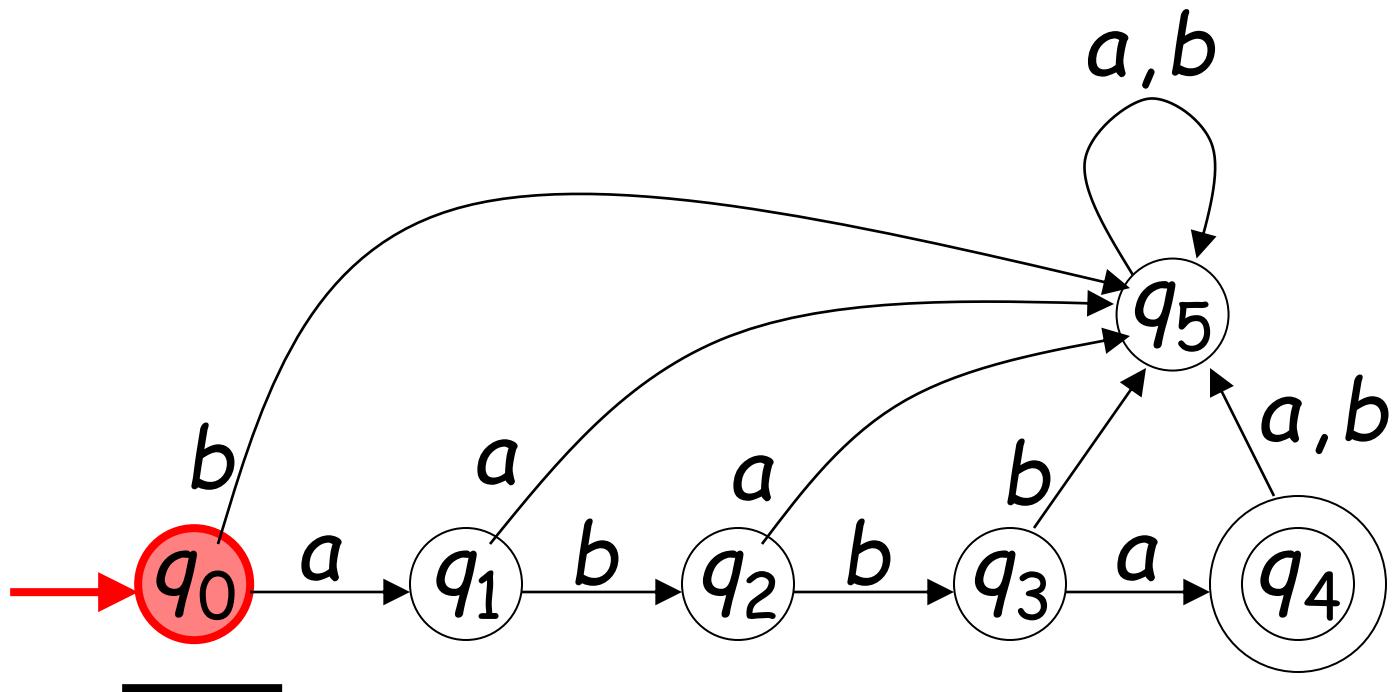
$Q$

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



# Initial State

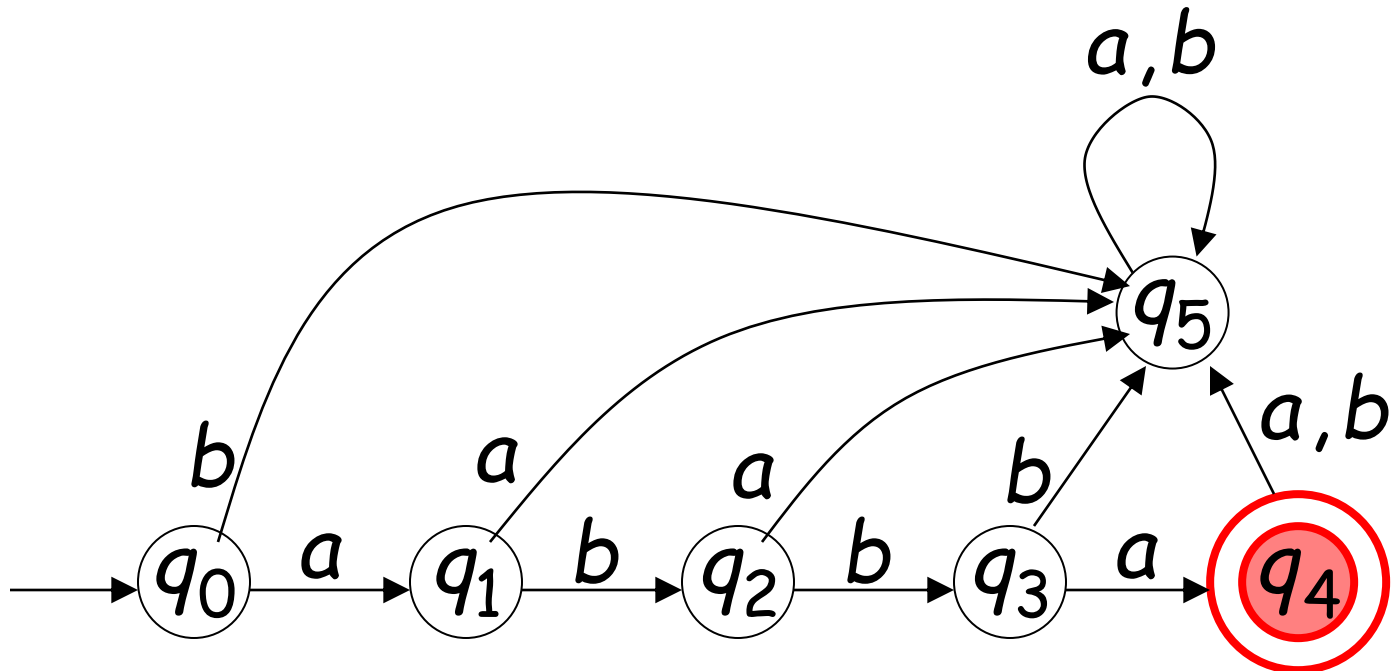
$q_0$





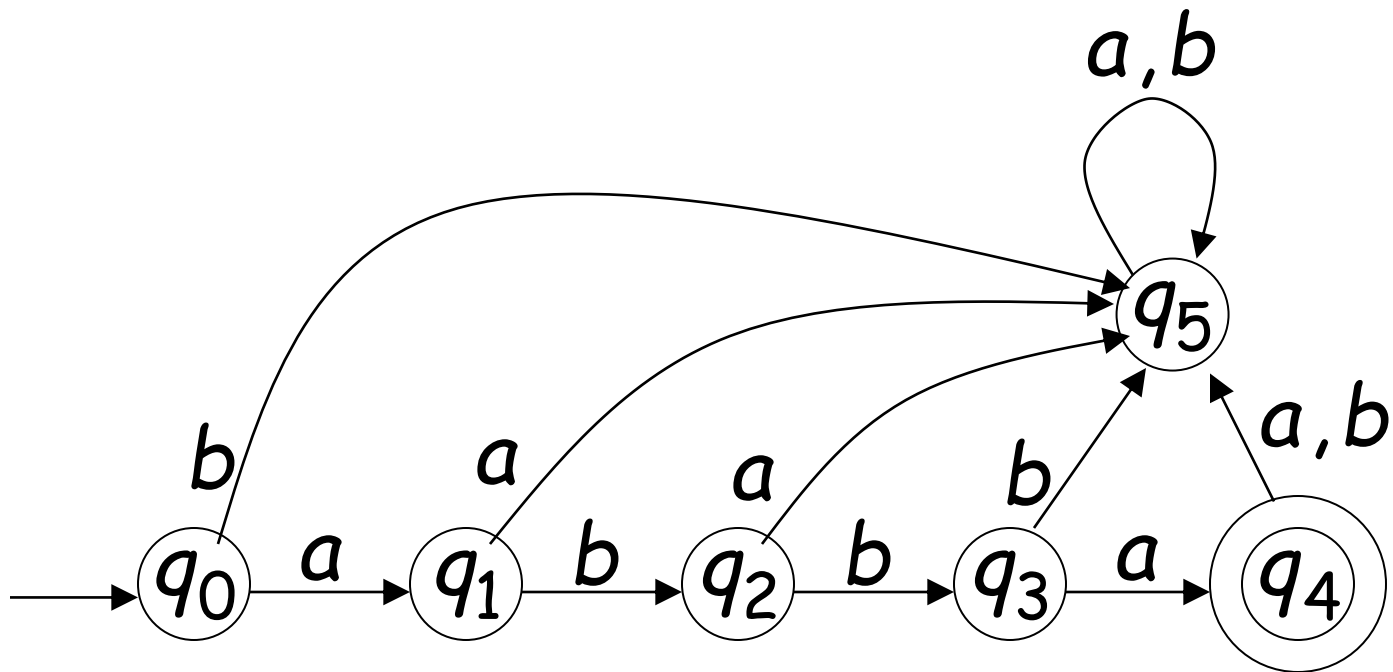
# Set of Accepting States or Final State $F$

□  $F = \{q_4\}$

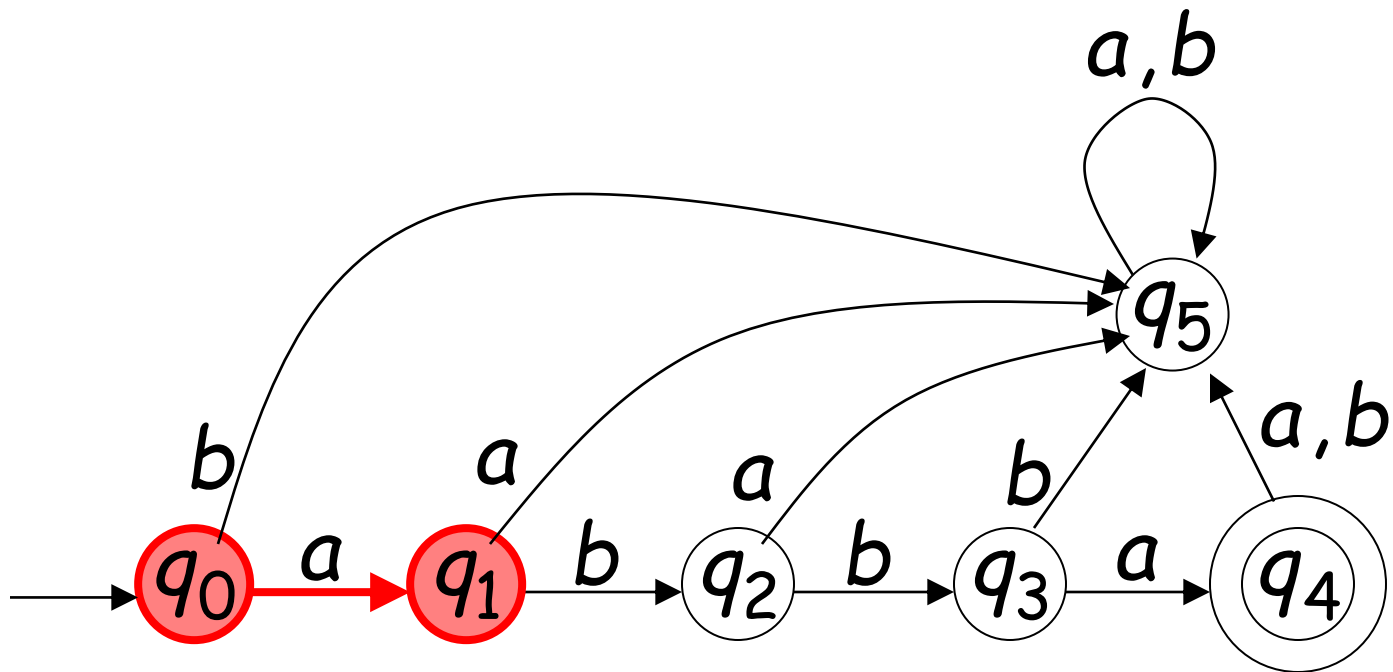


# Transition Function $\delta$

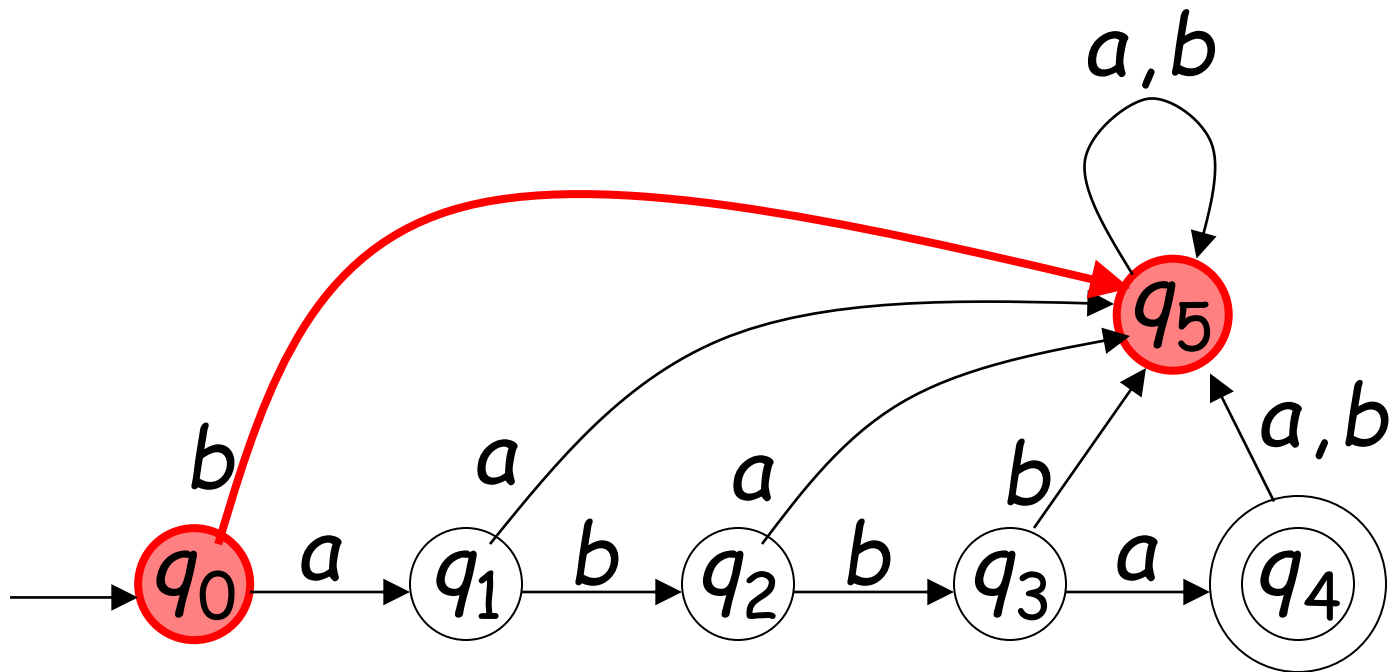
$$\delta : Q \times \Sigma \rightarrow Q$$



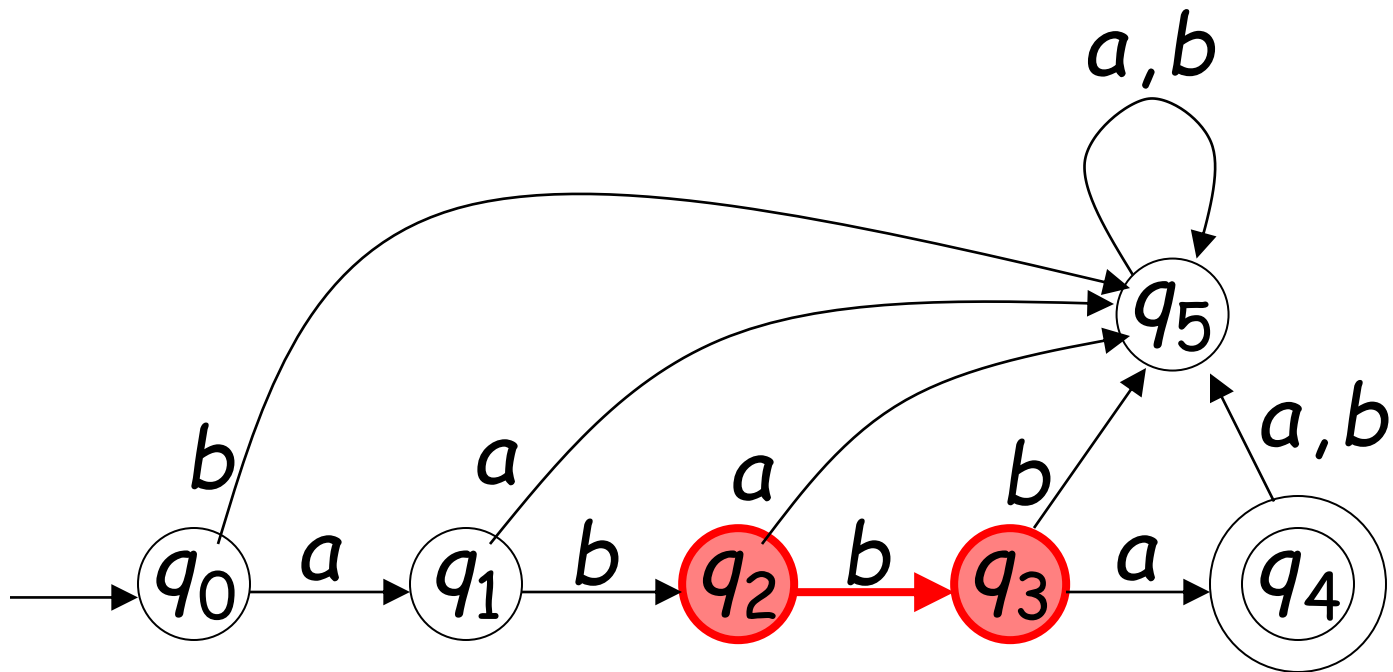
$$\delta(q_0, a) = q_1$$



$$\delta(q_0, b) = q_5$$



$$\delta(q_2, b) = q_3$$

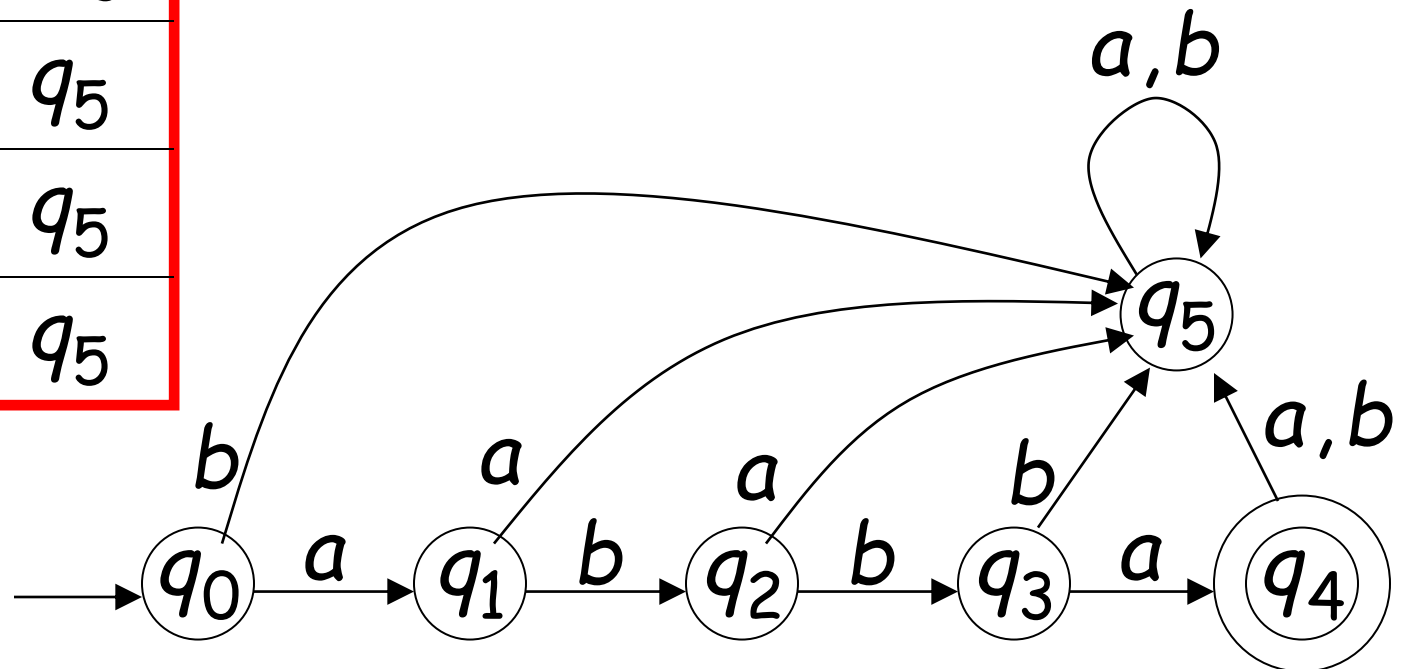


# Transition Function ( $\delta$ )

Contd...



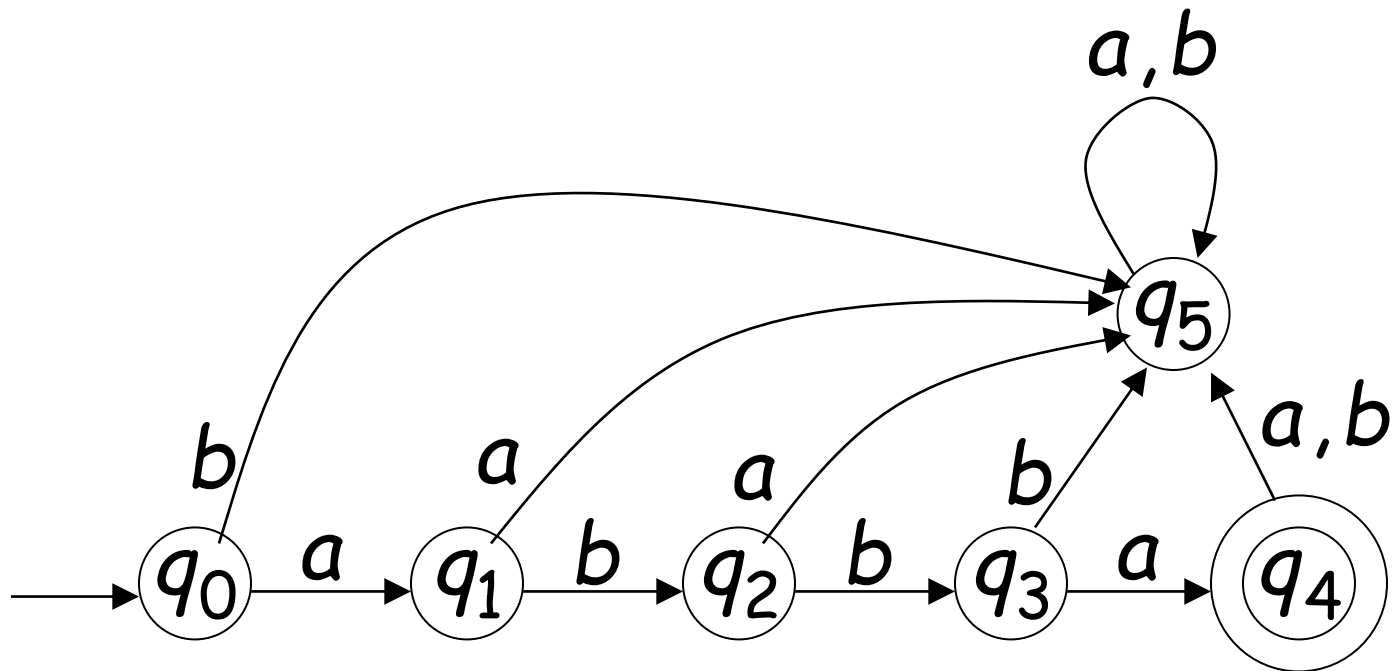
| $\delta$ | $a$   | $b$   |
|----------|-------|-------|
| $q_0$    | $q_1$ | $q_5$ |
| $q_1$    | $q_5$ | $q_2$ |
| $q_2$    | $q_5$ | $q_3$ |
| $q_3$    | $q_4$ | $q_5$ |
| $q_4$    | $q_5$ | $q_5$ |
| $q_5$    | $q_5$ | $q_5$ |



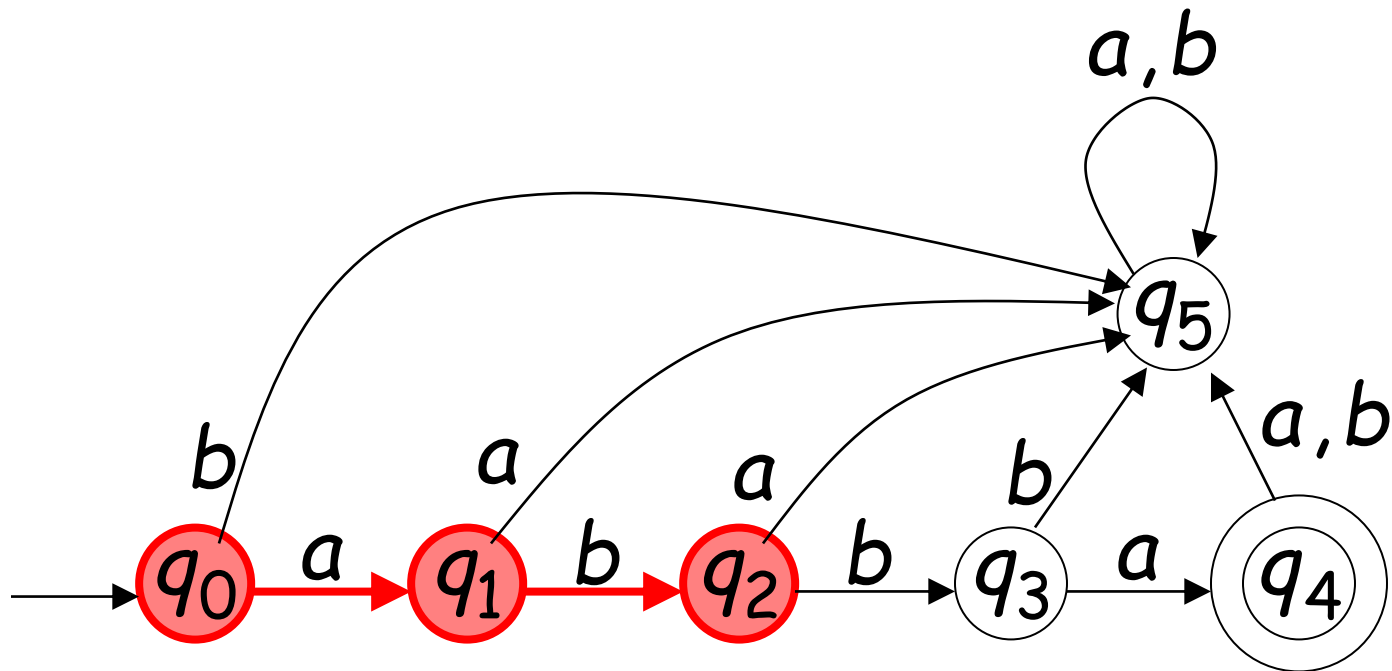
# Extended Transition Function $\delta^*$



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

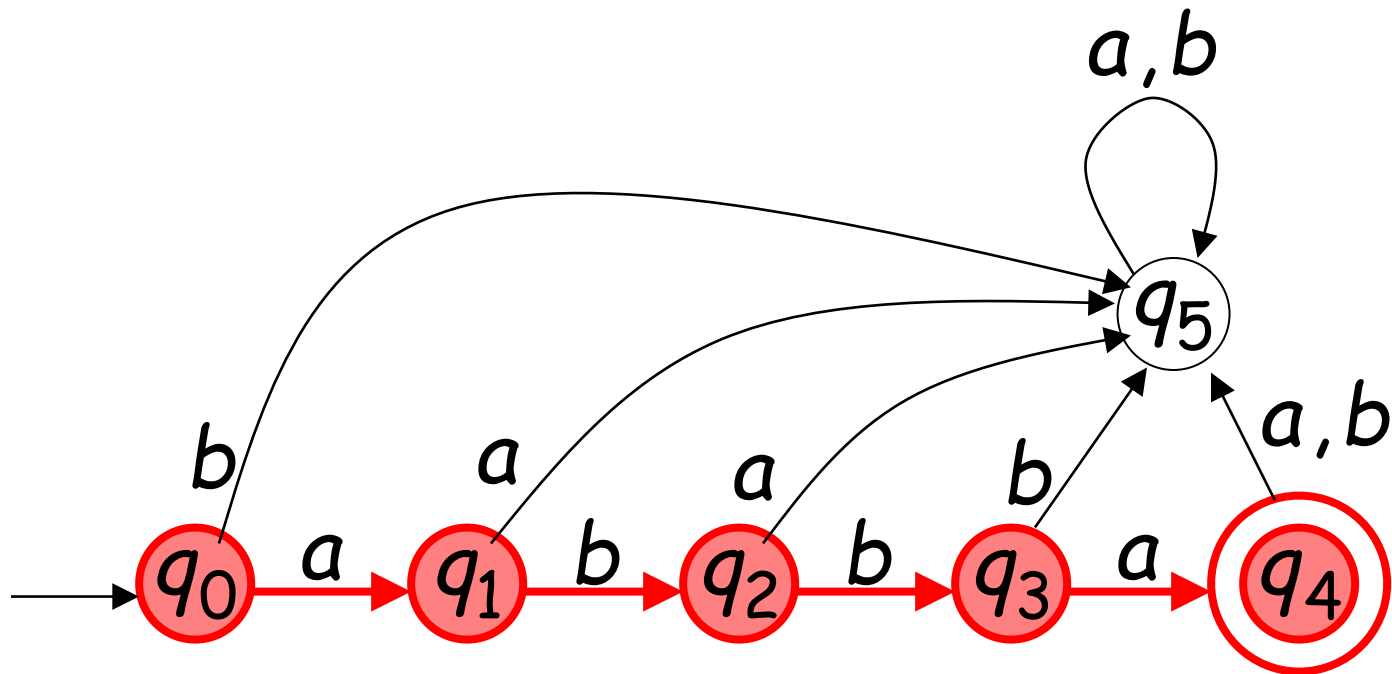


$$\delta^*(q_0, ab) = q_2$$

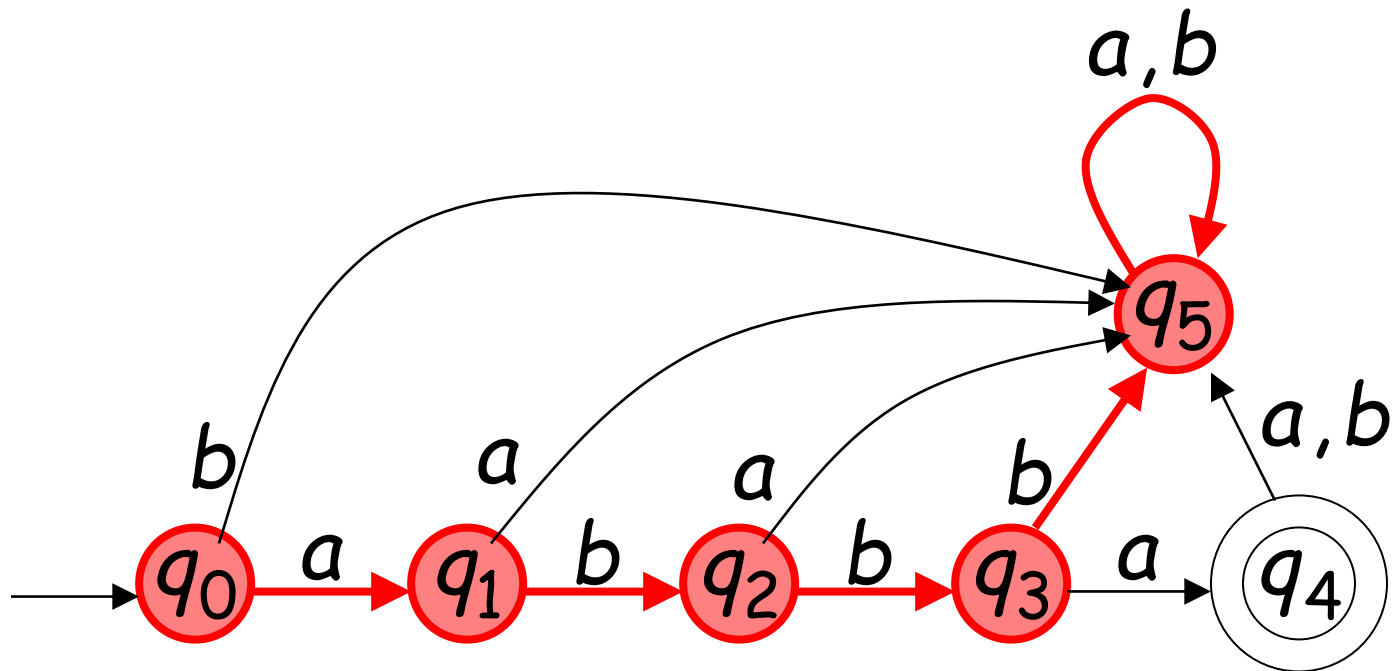




$$\delta^*(q_0, abba) = q_4$$



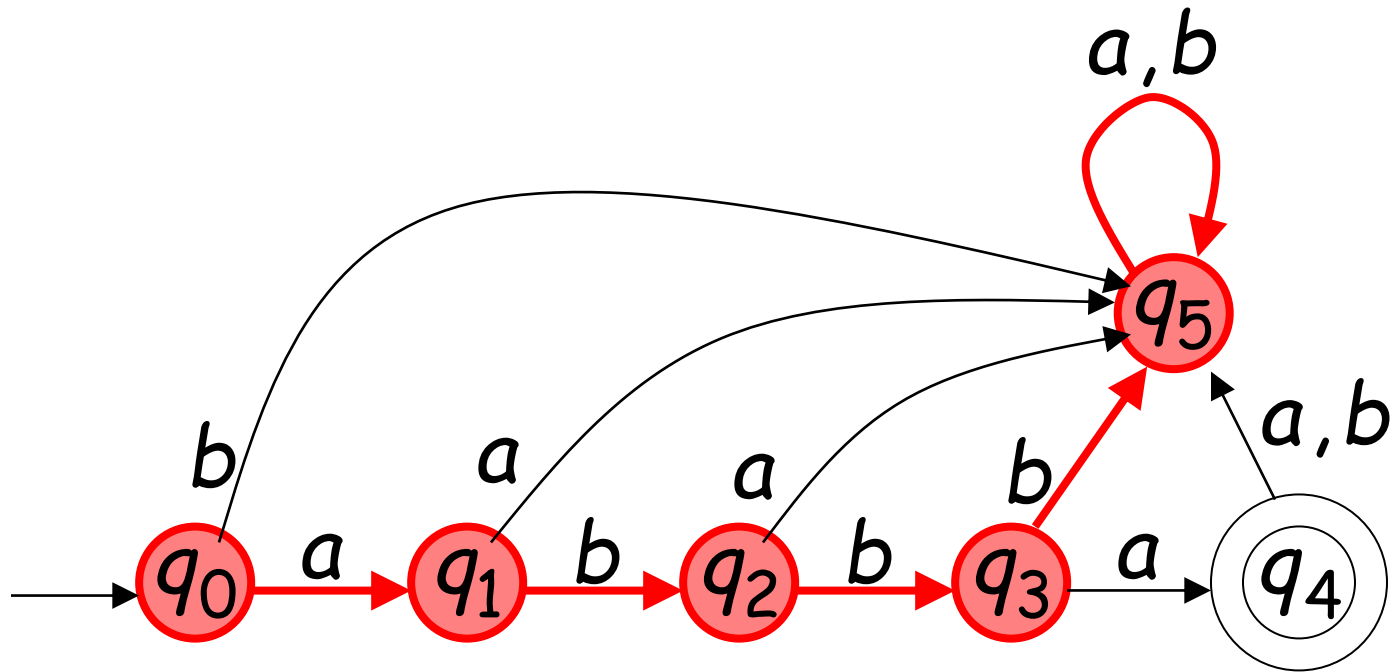
$$\delta^*(q_0, abbbaa) = q_5$$



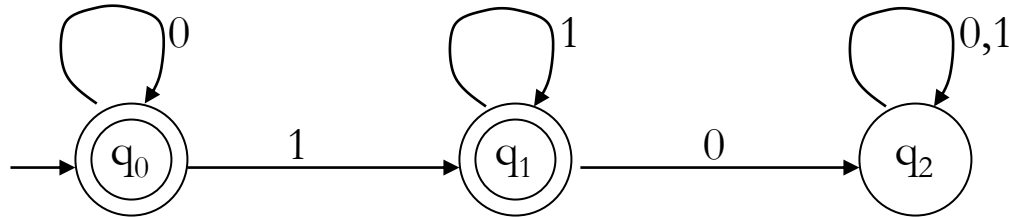
# Observation:

**Example:** There is a walk from  $q_0$  to  $q_5$   
with label  $abbbaa$

$$\delta^*(q_0, abbbaa) = q_5$$



# Example-2



alphabet  $\Sigma = \{0, 1\}$

states  $Q = \{q_0, q_1, q_2\}$

initial state  $q_0$

Final/ accepting states  $F = \{q_0, q_1\}$

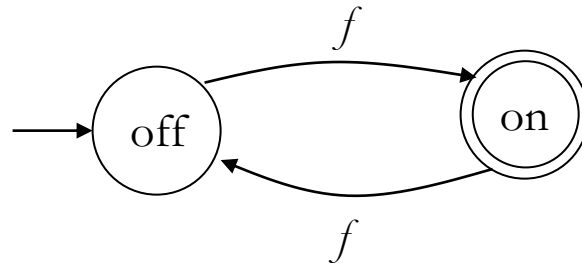
transition function  $\delta$ :

|        |       | inputs |       |
|--------|-------|--------|-------|
|        |       | 0      | 1     |
| states | $q_0$ | $q_0$  | $q_1$ |
|        | $q_1$ | $q_2$  | $q_1$ |
|        | $q_2$ | $q_2$  | $q_2$ |

# Applications

- *Vending Machine*
- *Traffic lights*
- *Video games*
- *Text parsing*
- *Protocol analysis*
- *Natural language processing*

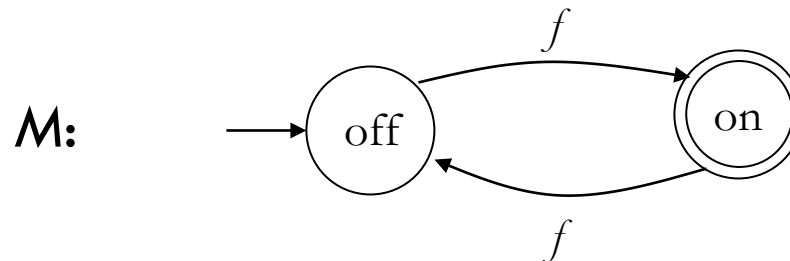
# Example of a finite automaton



- There are **states** off and on, the automaton **starts** in off and tries to reach the “**good state**” on
- What sequences of  $f$ s lead to the final state?
- Answer:  $\{f, fff, fffff, \dots\} = \{f^n : n \text{ is odd}\}$
- This is an **example** of a deterministic finite automaton over alphabet  $\{f\}$

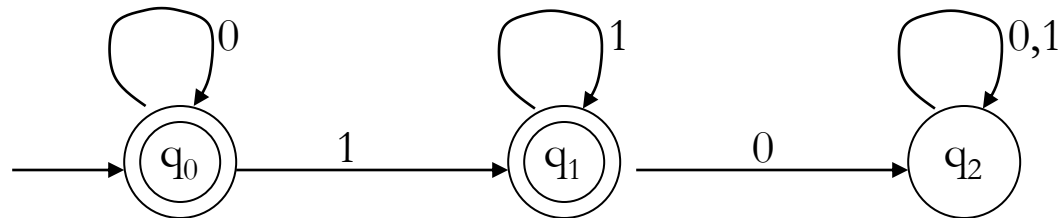
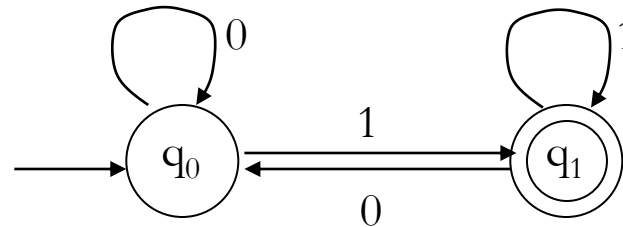
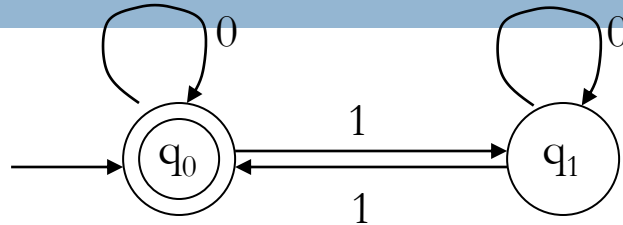
# Language of a DFA

The **language of a DFA**  $(Q, \Sigma, \delta, q_0, F)$  is the set of all strings over  $\Sigma$  that, starting from  $q_0$  and following the transitions as the string is read left to right, will reach some accepting state.



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

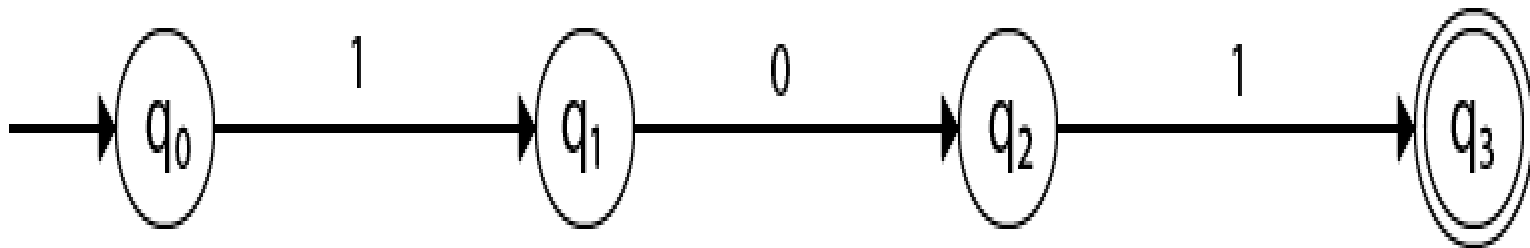
# Examples



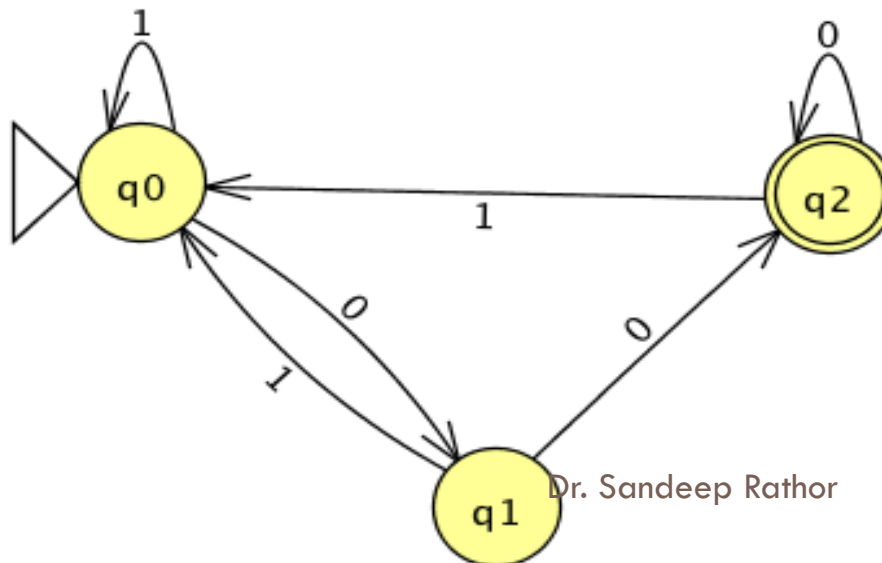
What are the languages of these DFAs?



**Example:** Design a FA with  $\Sigma = \{0, 1\}$   
accepts the only input 101.



**Example:** The set of all strings  $\Sigma = \{0, 1\}$  ending  
in 00

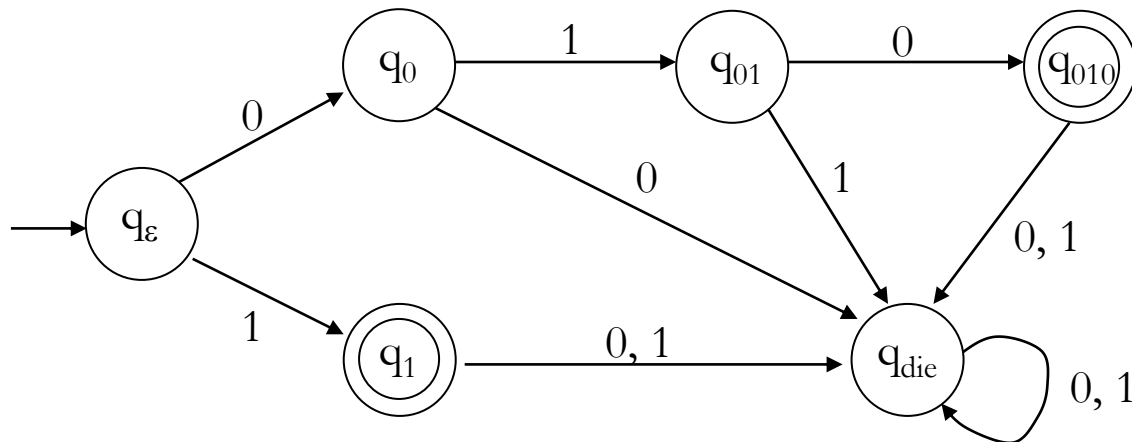


# Examples

- Construct a DFA that accepts the language

$$L = \{010, 1\} \quad (\Sigma = \{0, 1\})$$

- Answer



# $\Sigma^+$ and $\Sigma^*$

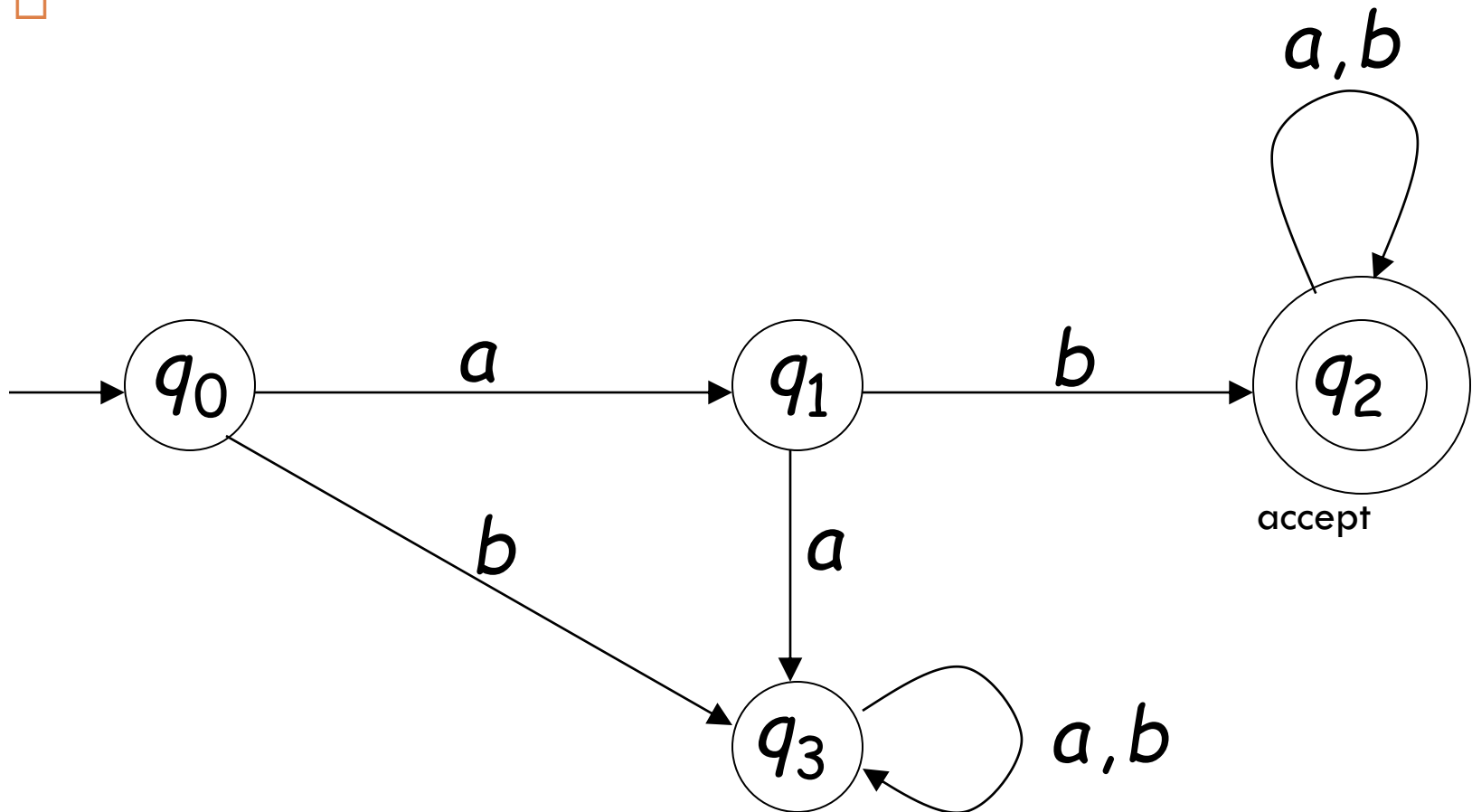


- $\Sigma$  is an alphabet ,  $\Sigma = \{y\}$
- $\Sigma^*$  is the set of all strings including null or obtained by concatenating zero or more symbols from  $\Sigma$
- $\Sigma^* = \{\epsilon, y, yy, yyy, yyyy, \dots\}$
  
- $\Sigma^+$  is the set of all strings excluding null or obtained by concatenating one or more symbols
- $\Sigma = \{y\}$
- $\Sigma^+ = \{y, yy, yyy, yyyy, \dots\}$

# Example-4: Design a DFA for

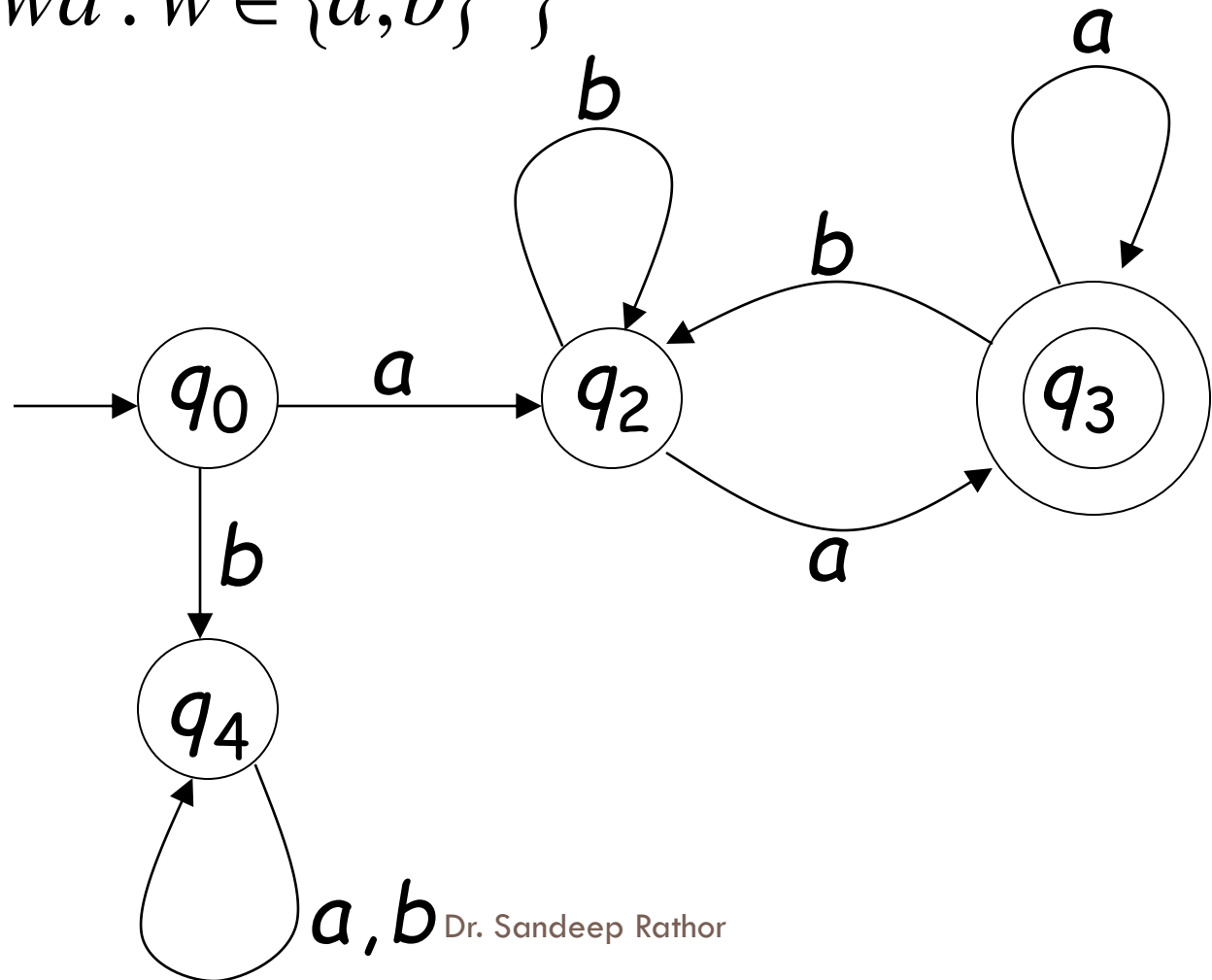


$L(M) = \{ \text{all strings with prefix } ab \}$



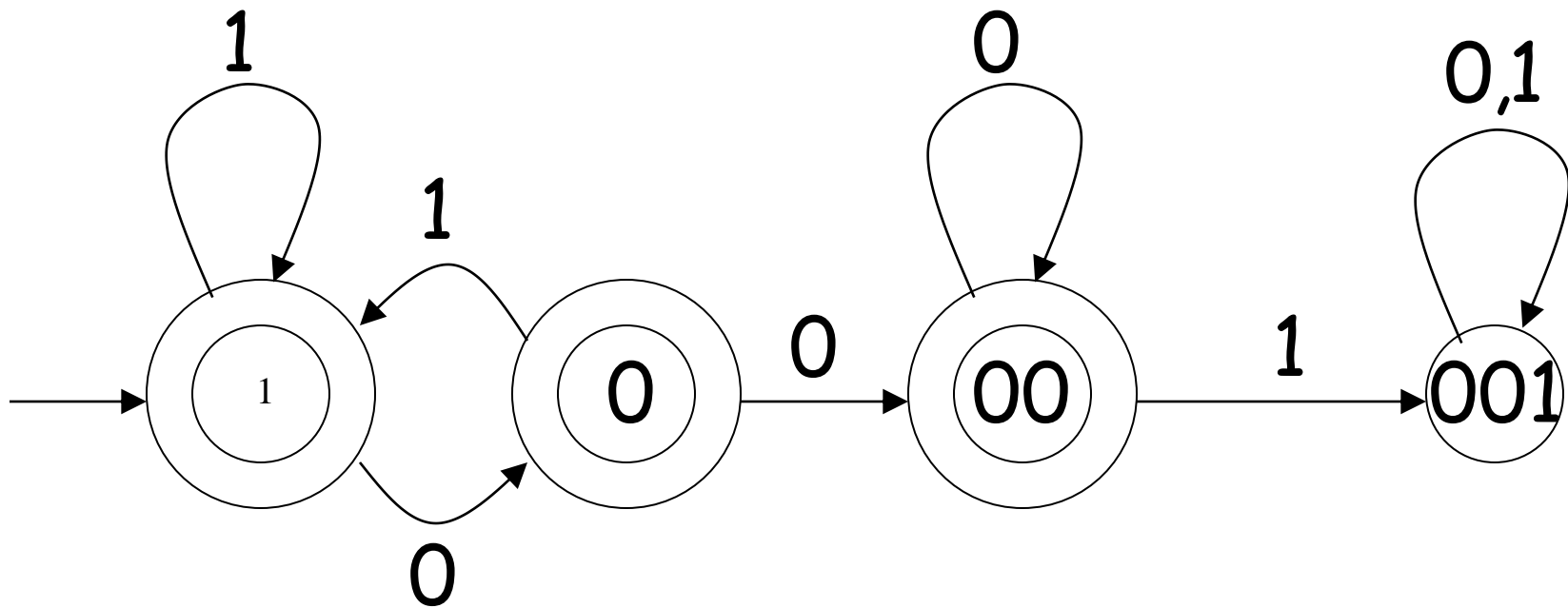
# Example-5: DFA for

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



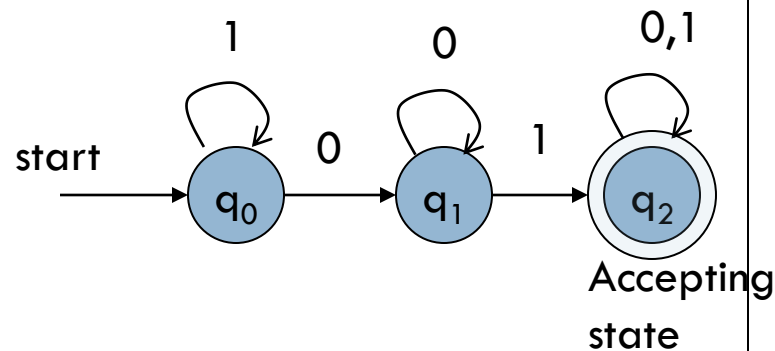
# Example-6: DFA for

$L(M) = \{ \text{all strings without substring } 001 \}$



**Ex-7:** Design a DFA for language:  $L = \{w \mid w \text{ is a binary string that contains } 01 \text{ as a substring}\}$

# DFA for strings containing 01



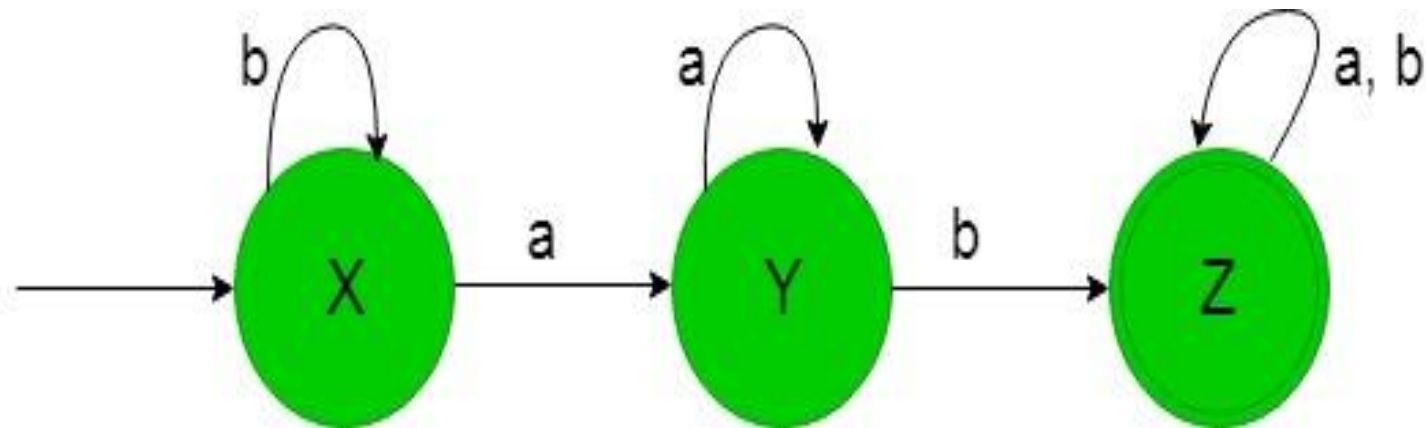
- What if the language allows empty strings?

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{0, 1\}$
- start state =  $q_0$
- $F = \{q_2\}$
- Transition table

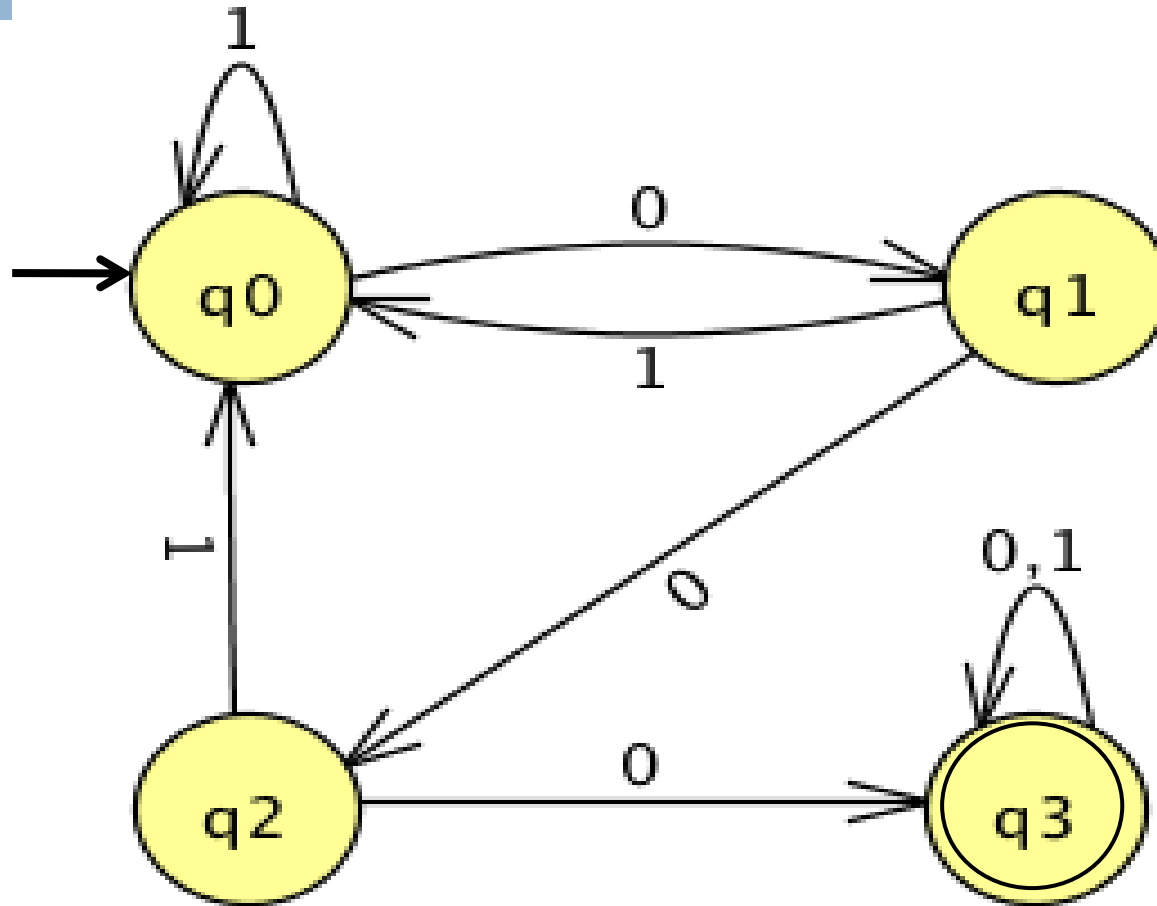
|          |        | symbols |       |
|----------|--------|---------|-------|
| $\delta$ |        | 0       | 1     |
| states   | $q_0$  | $q_1$   | $q_0$ |
|          | $q_1$  | $q_1$   | $q_2$ |
|          | $*q_2$ | $q_2$   | $q_2$ |



Construction of a DFA accepting set of string over  $\{a, b\}$  where each string containing 'ab' as the substring.

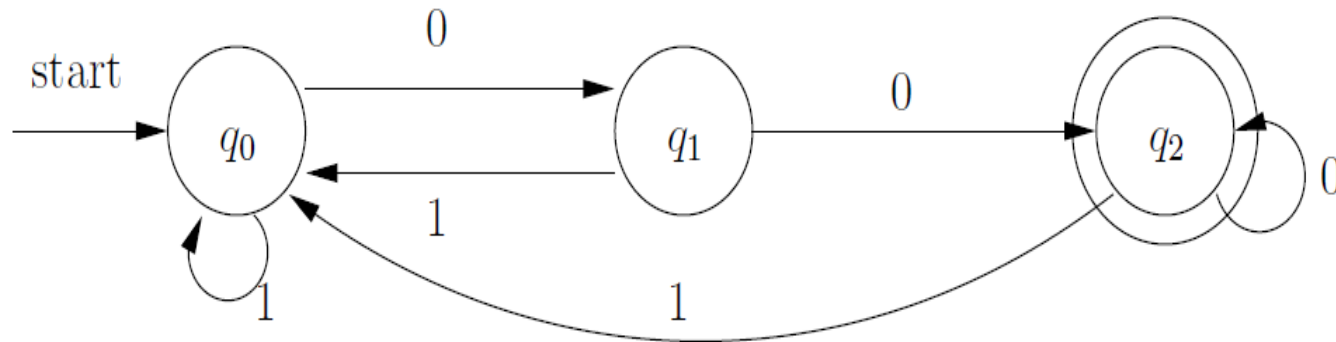


**Example:** The set of all strings with three consecutive 0's (not necessarily at the end).



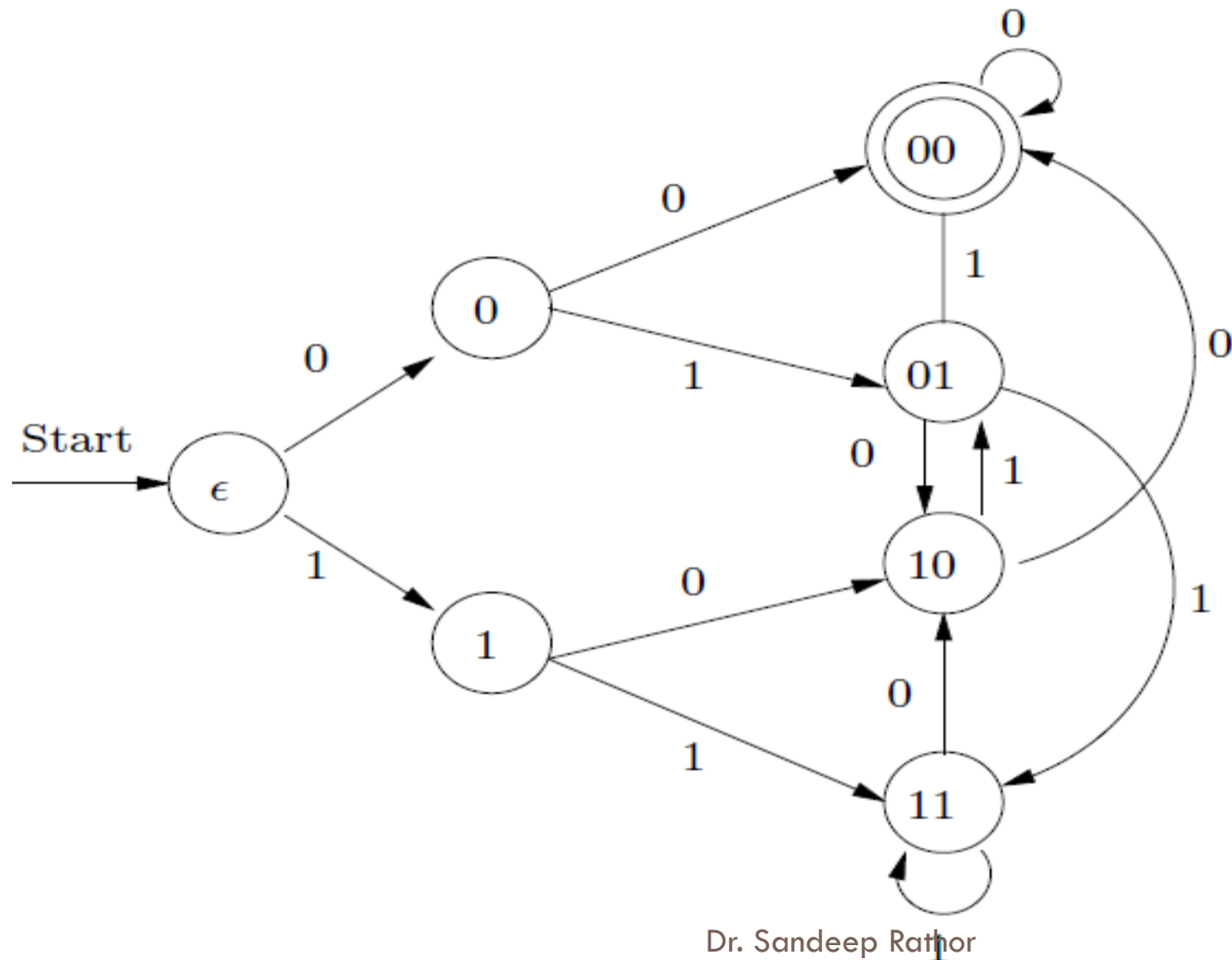
# Construct DFA's accepting the following languages over the alphabet $\{0, 1\}$ .

## 1. The set of all strings ending in 00.

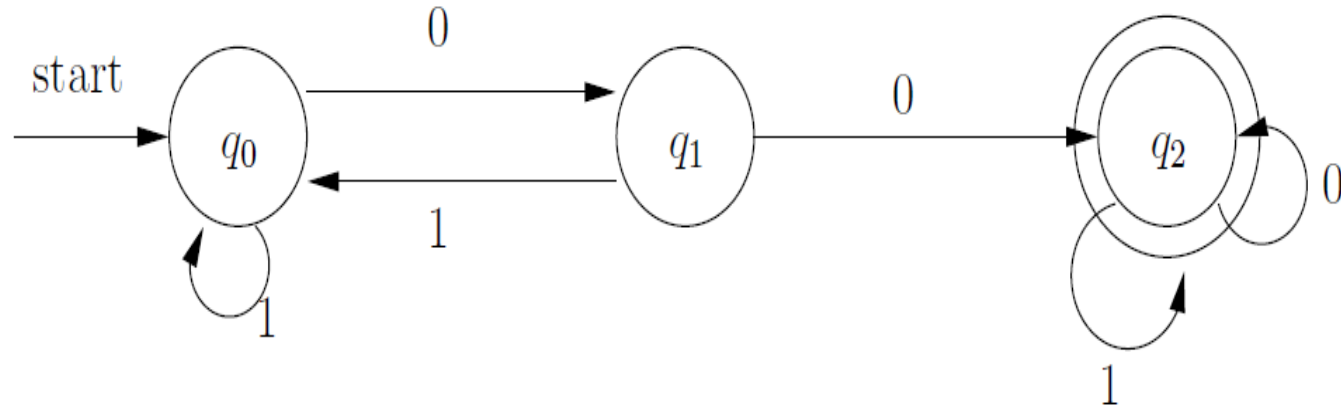


OR

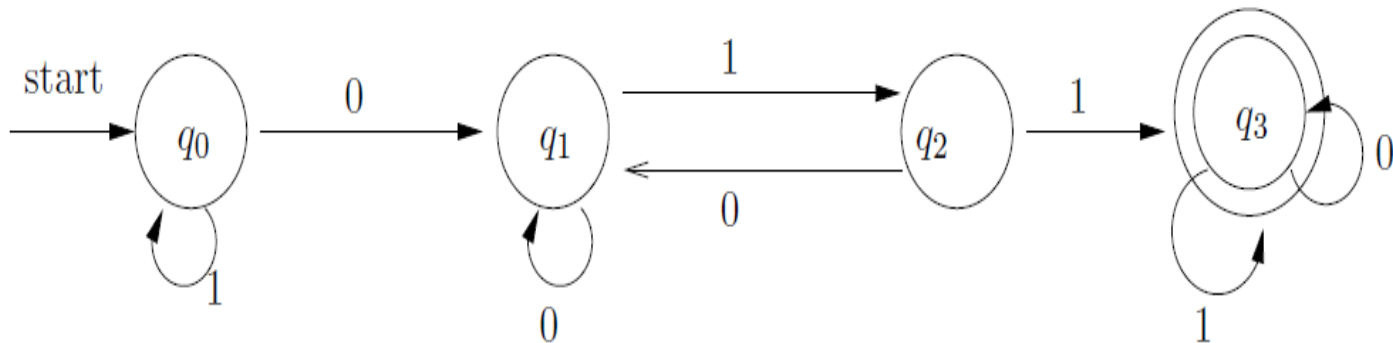
The set of all strings ending in 00.



2. The set of all strings with two consecutive 0's (not necessarily at the end).



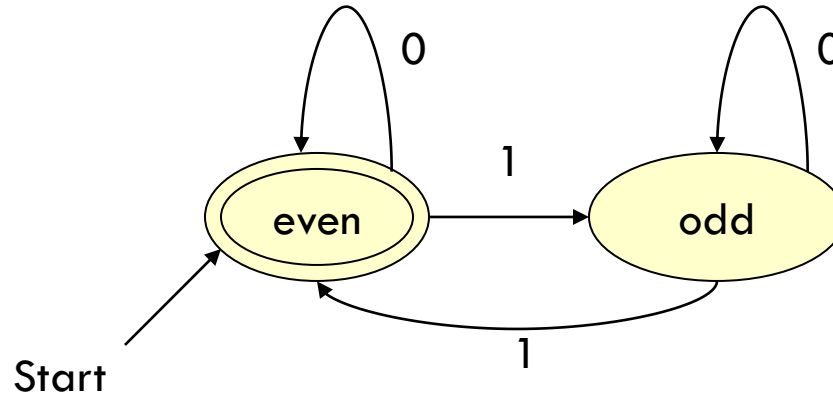
3. The set of strings with 011 as a substring



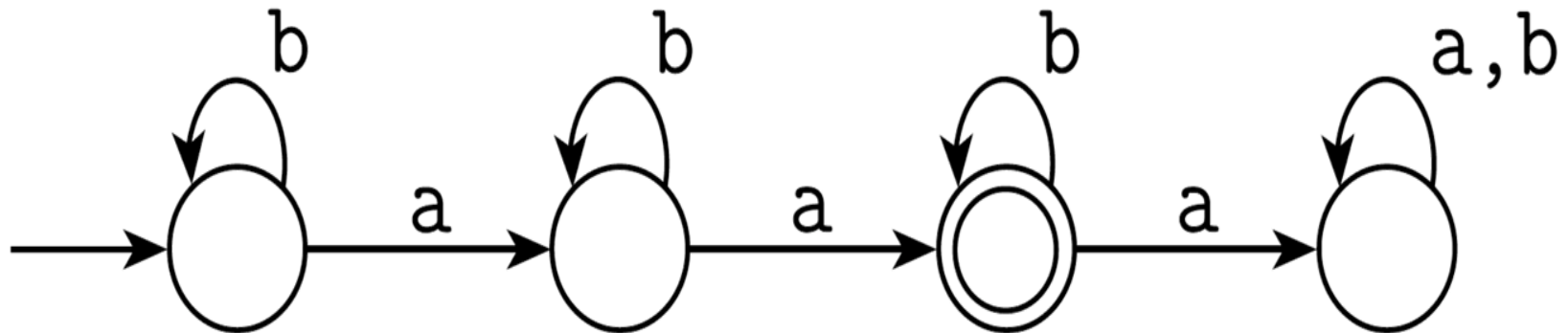
# Practice Questions...



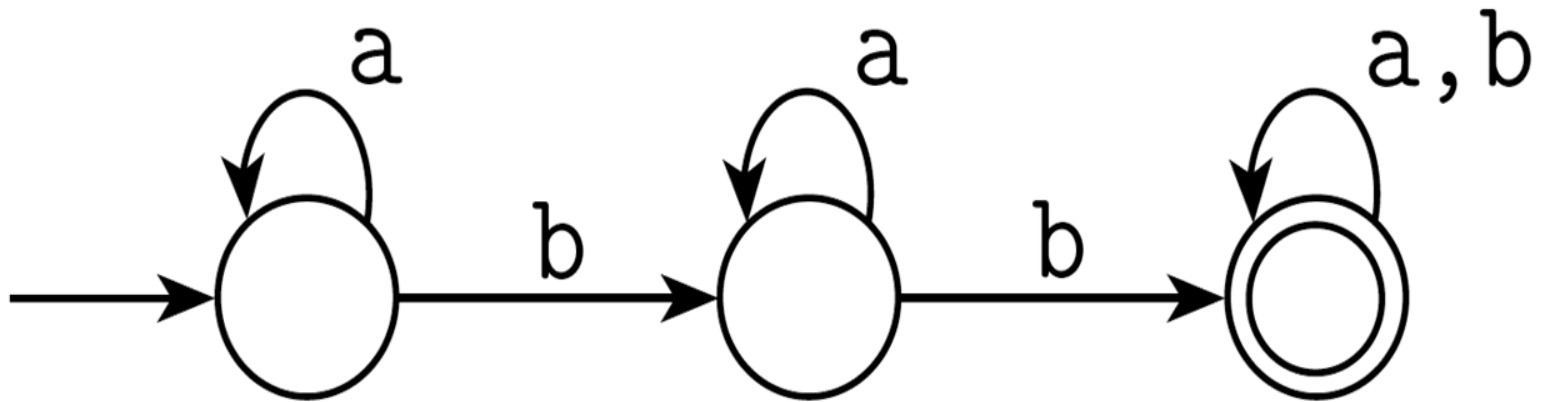
## DFA for: An Even Number of 1's



# Exactly Two a's

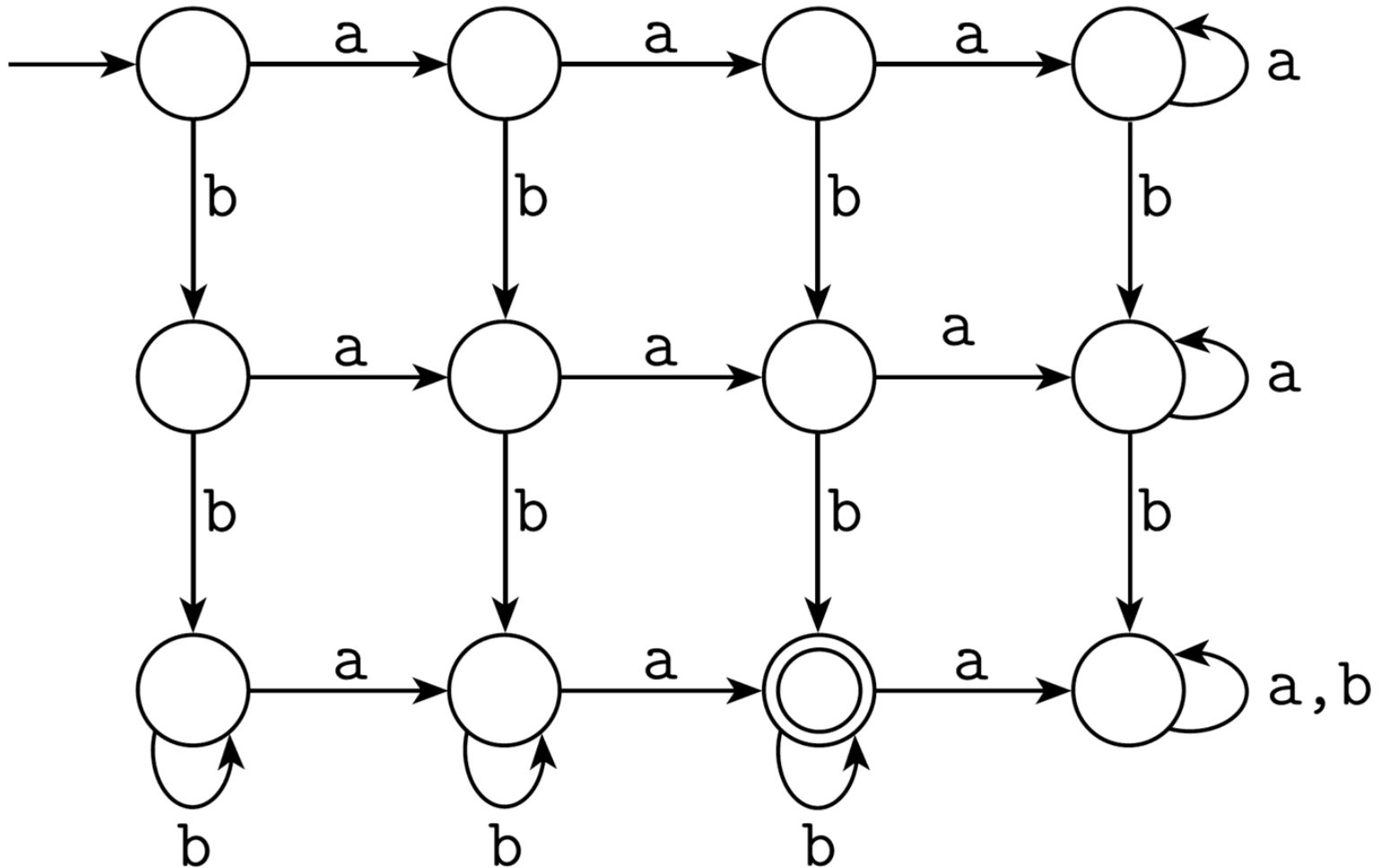


# At Least Two b's



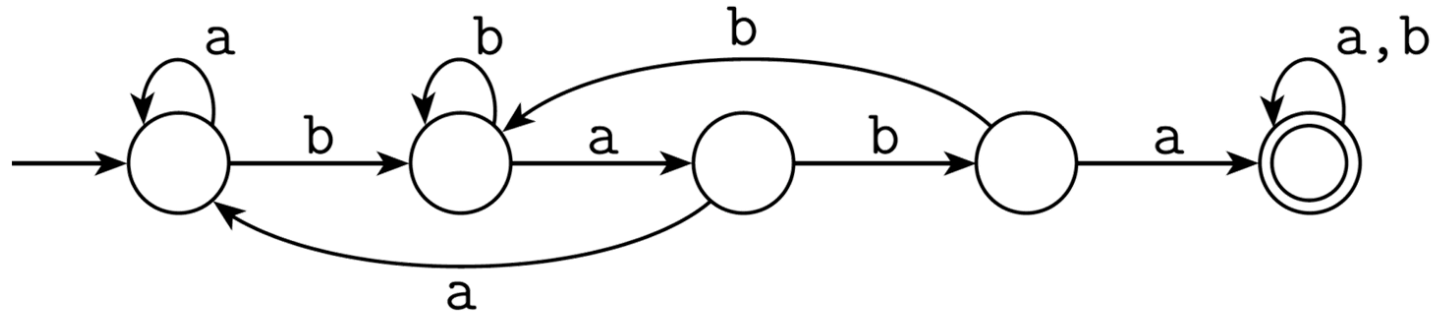


# Exactly two a's and at least two b's

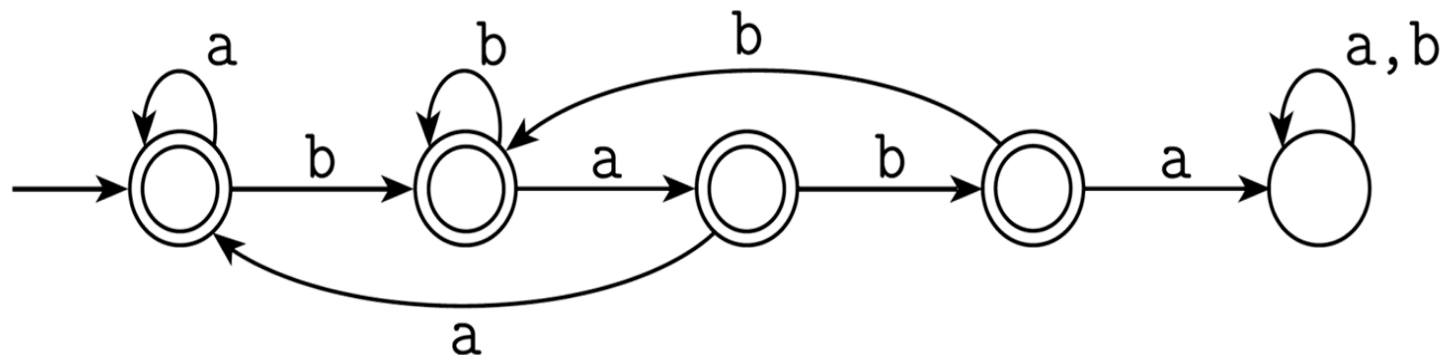


# Containing Substrings or Not

- Contains baba:



- Does not contain baba:



# Non-Deterministic Finite Automata (NFA)



A **Non-deterministic Finite Automata** (NFA) is a 5-tuple

$(Q, \Sigma, \delta, q_0, F)$  where:

- ▣  $Q$  is a finite set of **states**
- ▣  $\Sigma$  is an **input alphabet**
- ▣  $\delta$ : is a **transition function**  $Q \times \Sigma \rightarrow 2^Q$  [power set of  $Q$ ]
- ▣  $q_0 \in Q$  is an **initial state**
- ▣  $F \subseteq Q$  is a set of **accepting states** (or **final states**).

NFAs were introduced in 1959 by Michael O. Rabin and Dana Scott

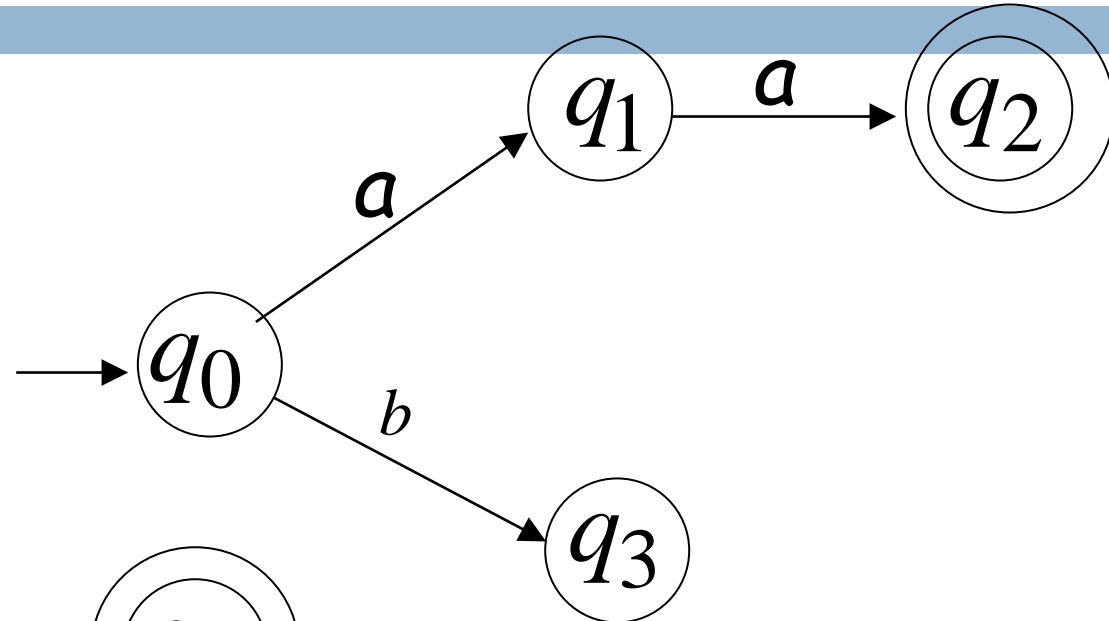
# Applications of NFA

---

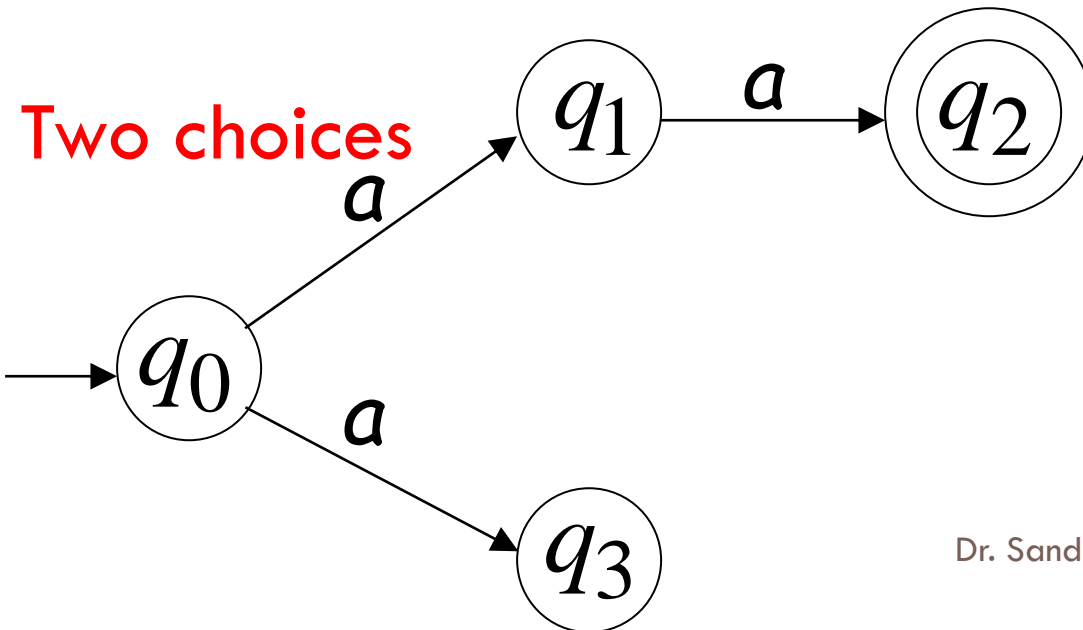
1. Chess
2. Tic Tac Toe
3. Ludo
4. Playing Card

# Example of DFA & NFA

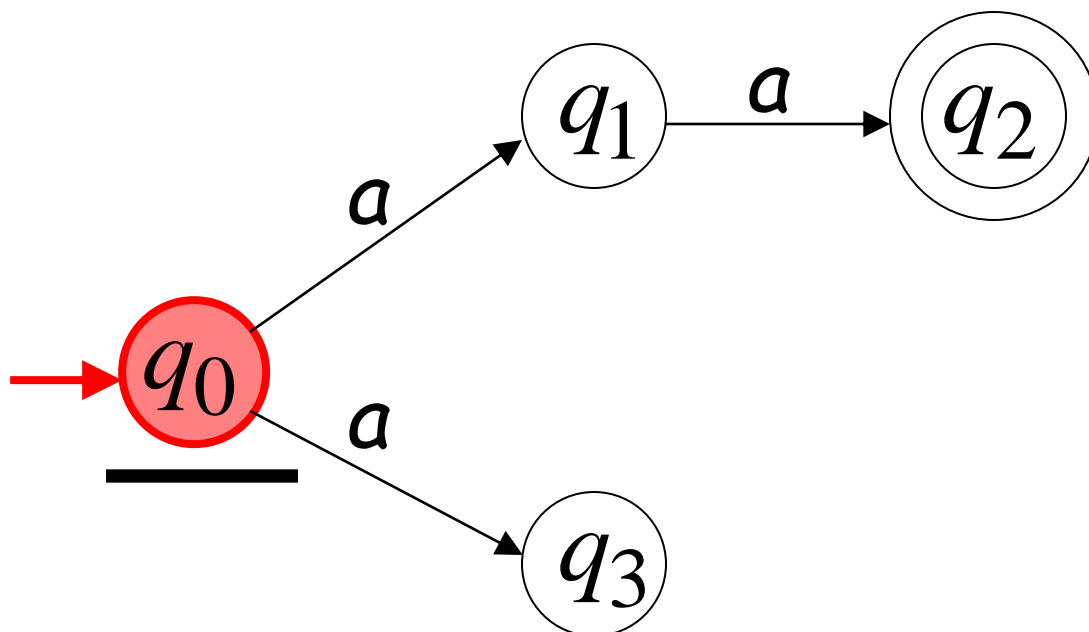
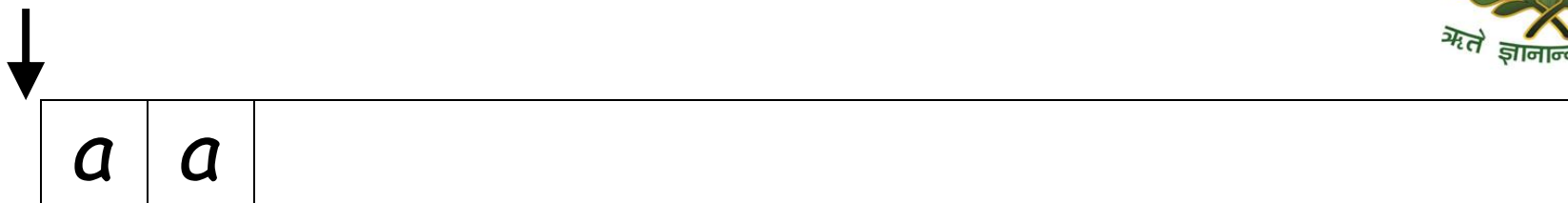
**DFA**



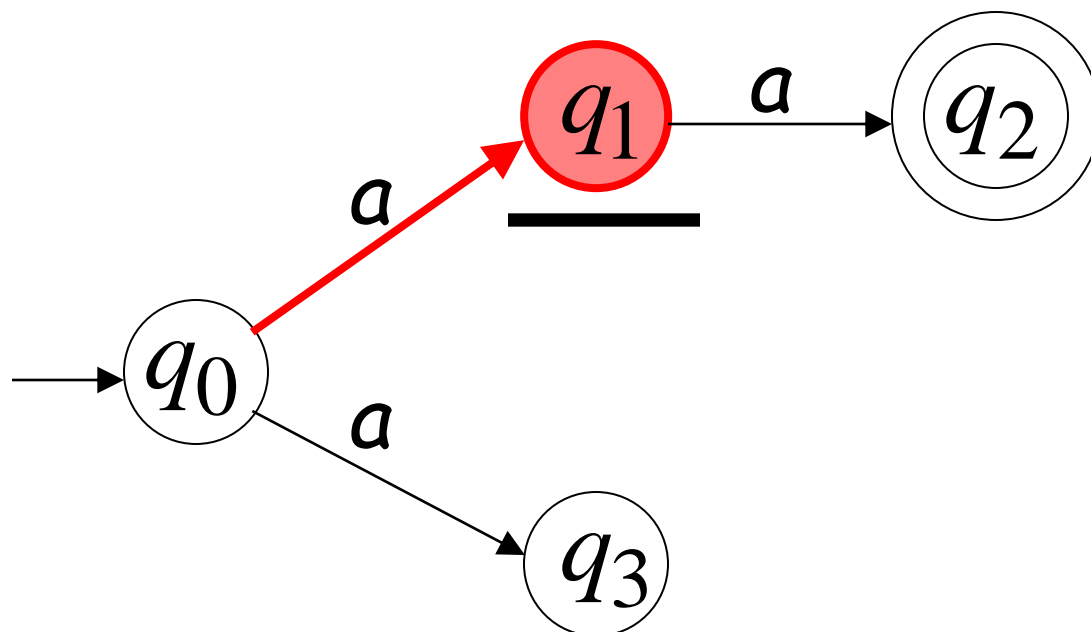
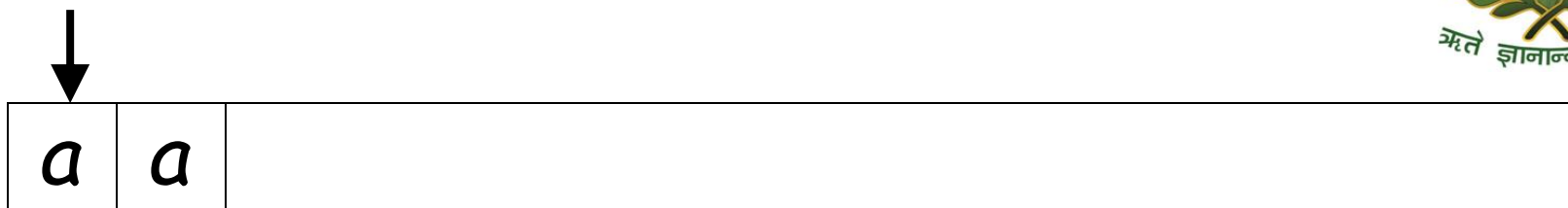
**NFA/ NDFA**



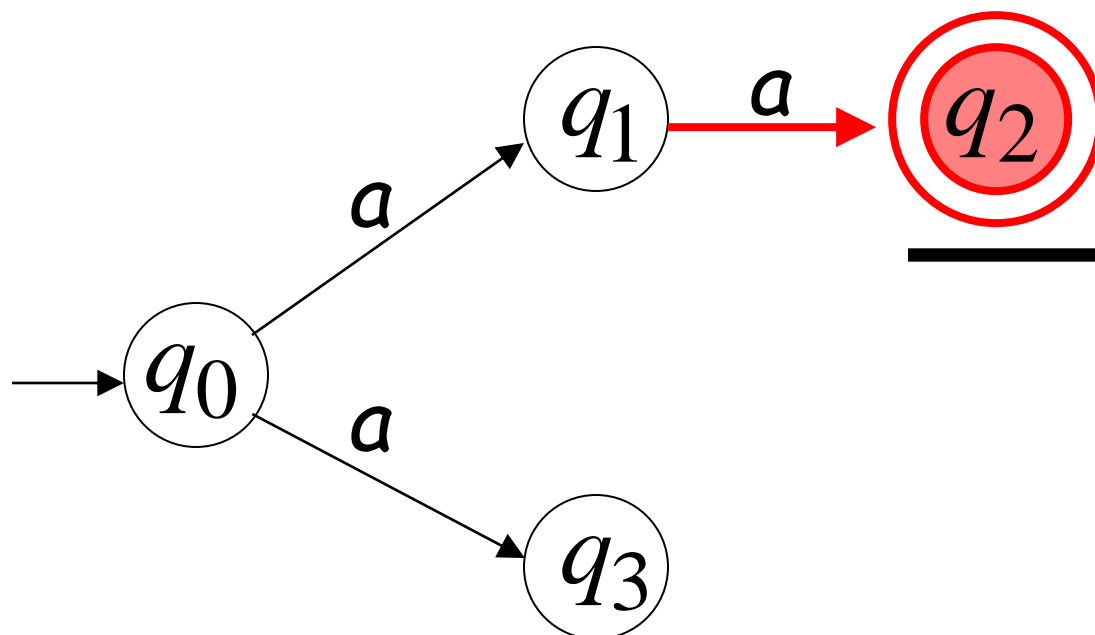
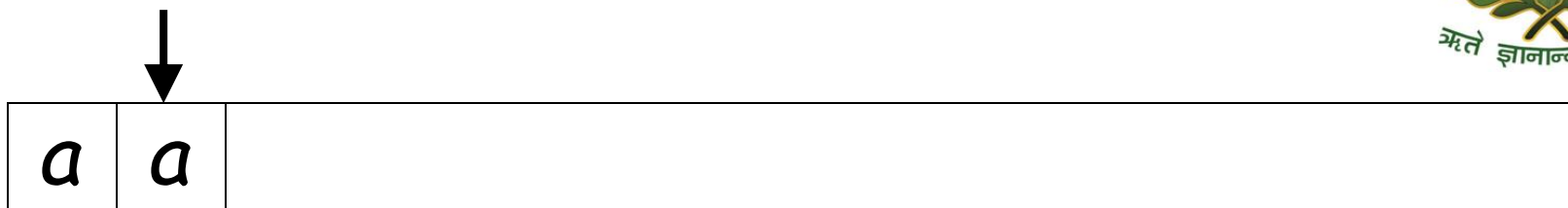
# First Choice



# First Choice

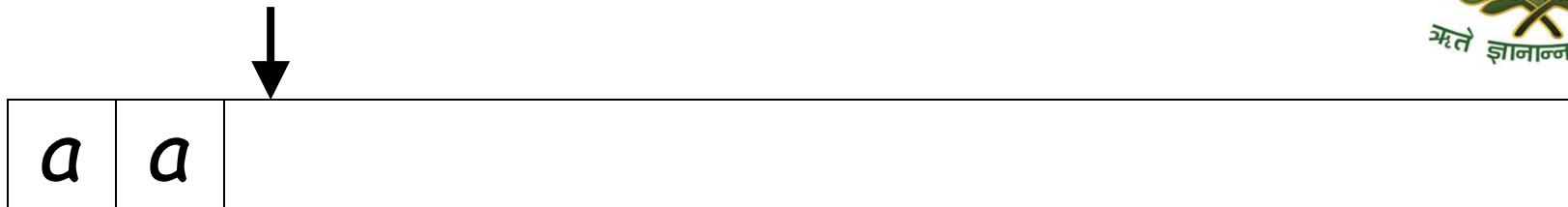


# First Choice

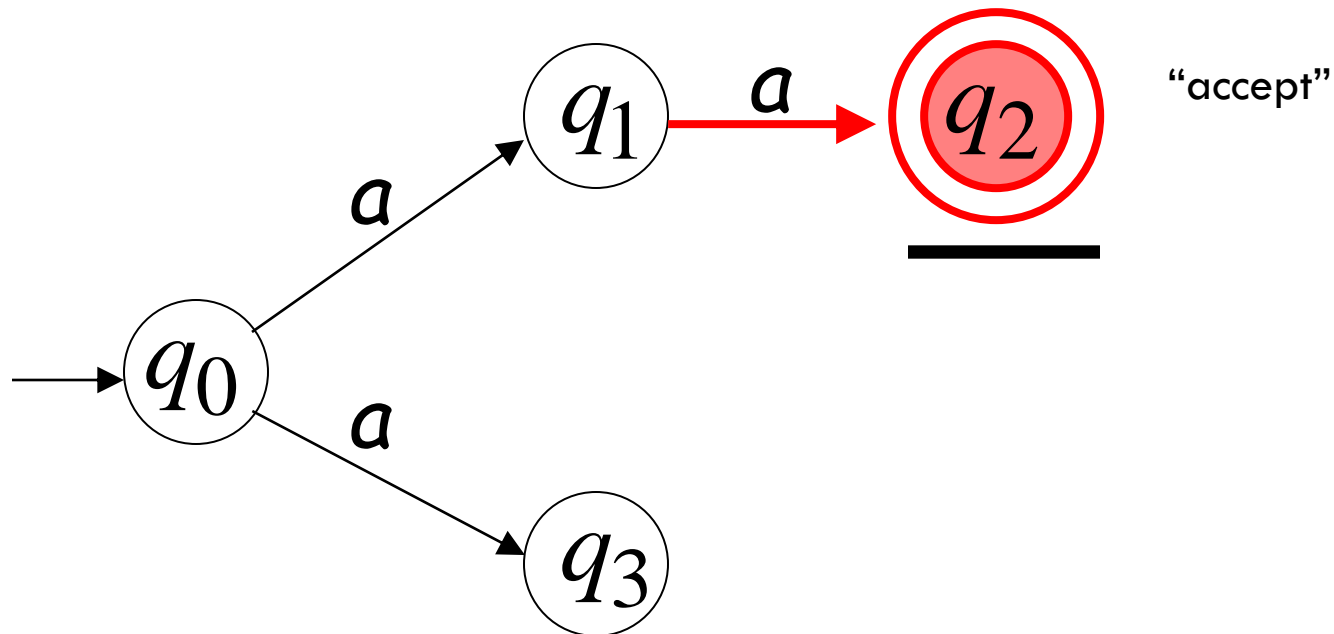




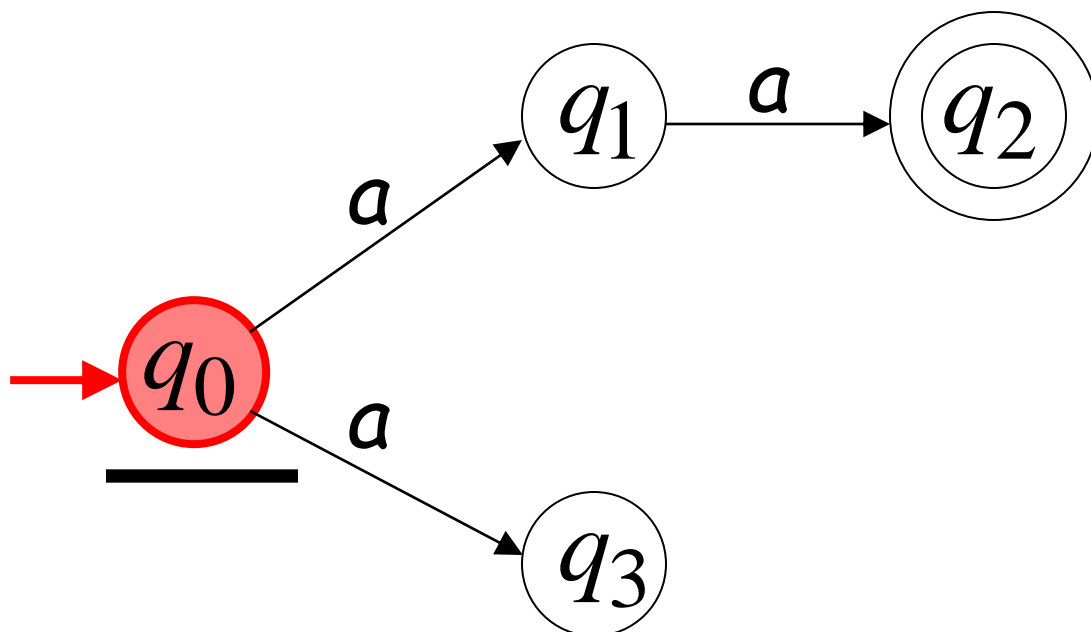
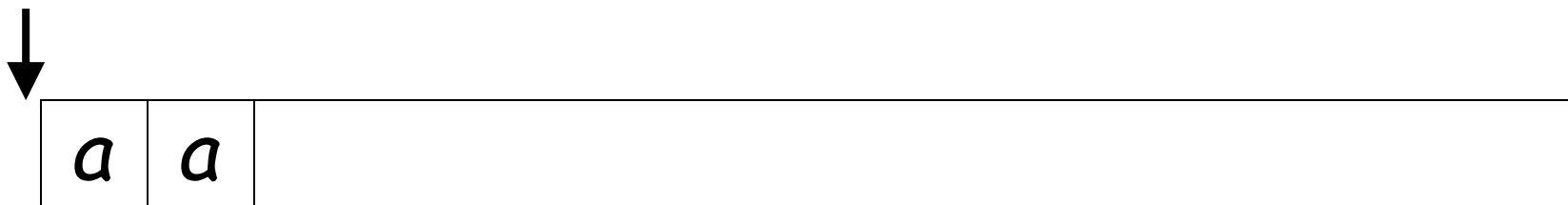
# First Choice



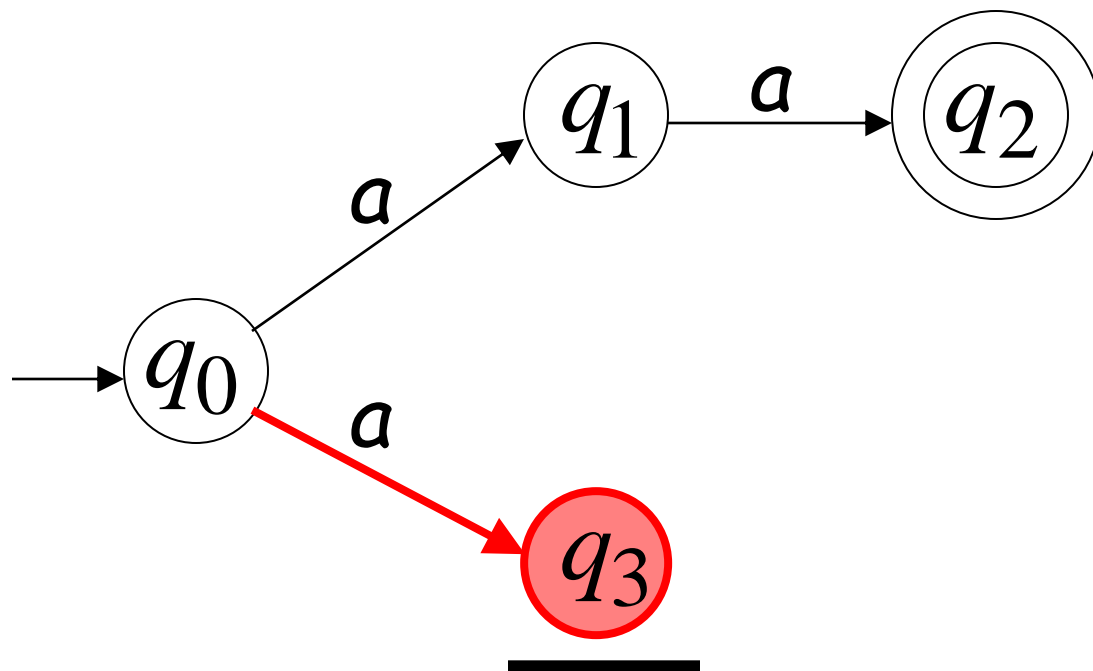
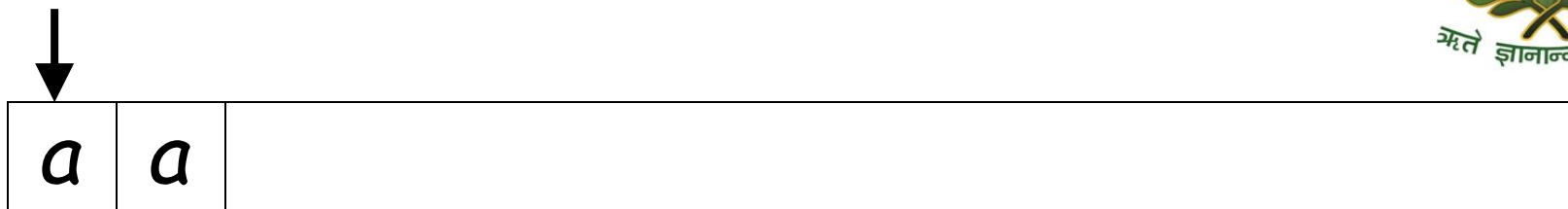
All input is consumed



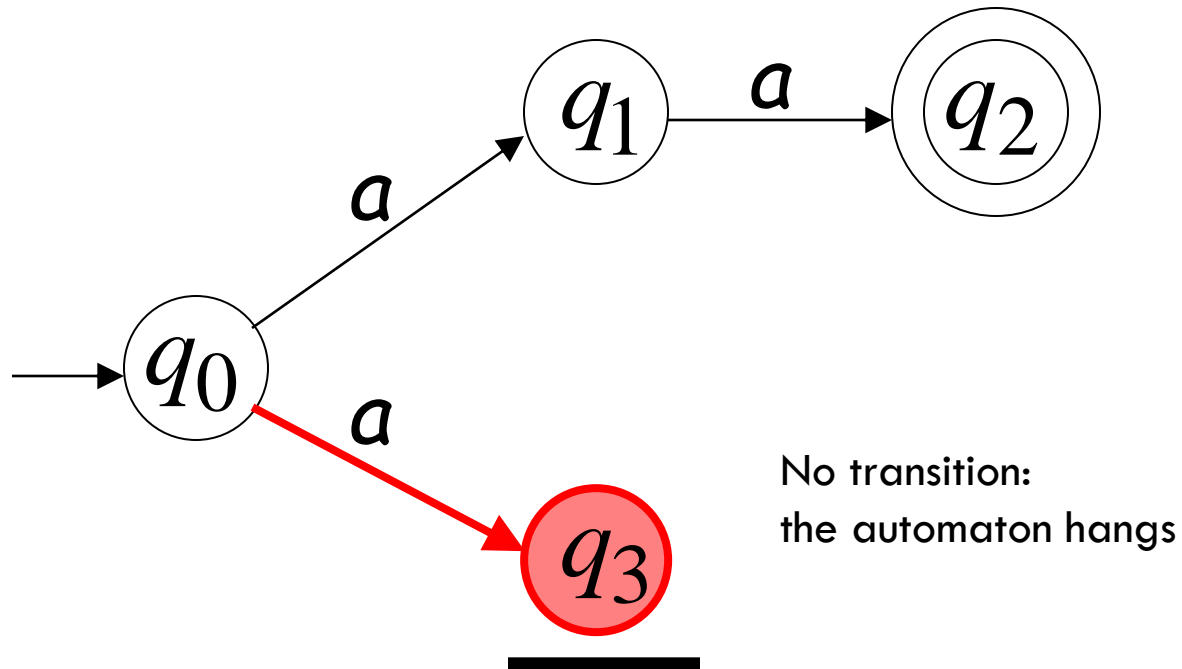
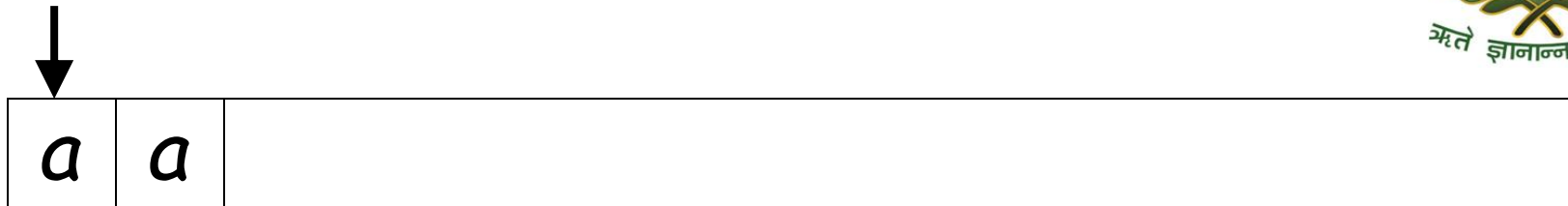
# Second Choice



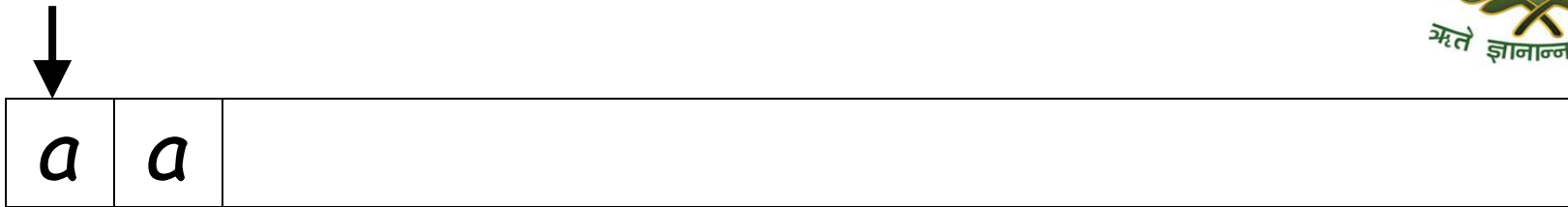
# Second Choice



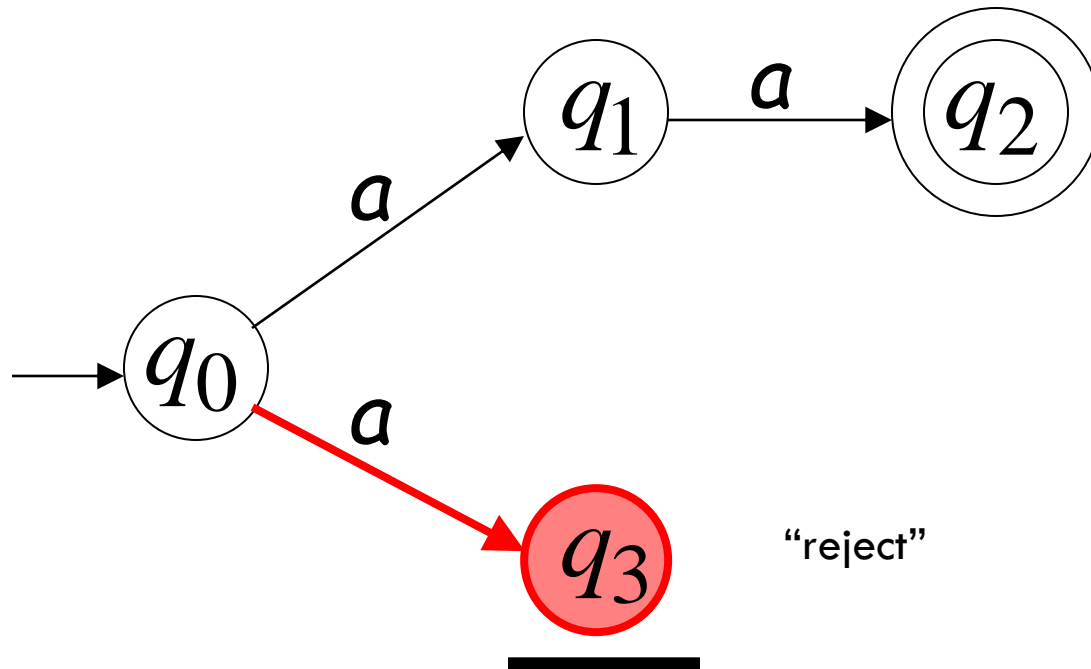
# Second Choice



# Second Choice

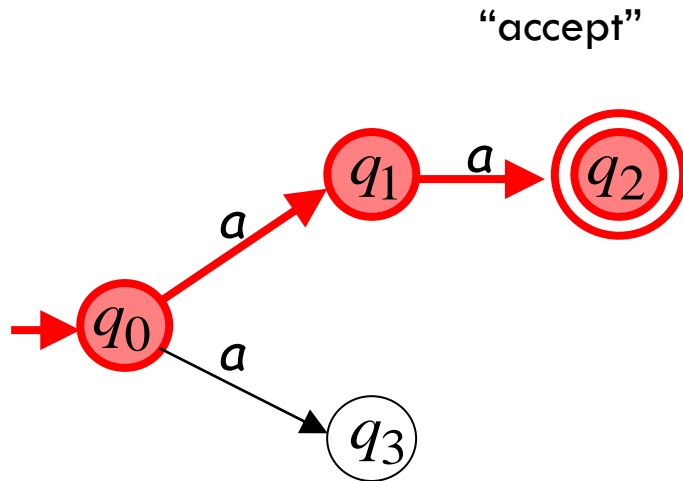


Input cannot be consumed



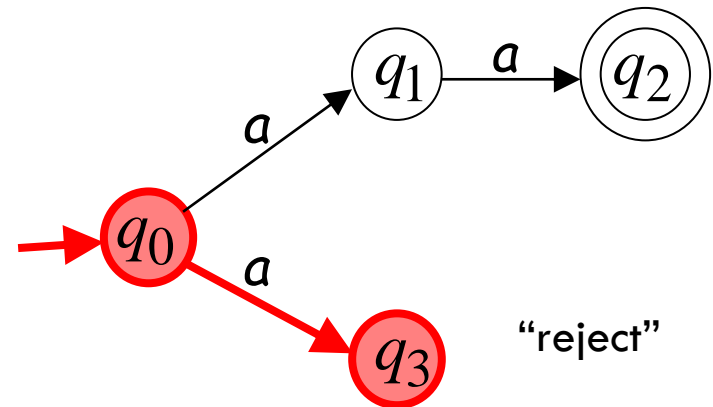
# Example

*aa* is accepted by the NFA:



because this  
computation  
accepts

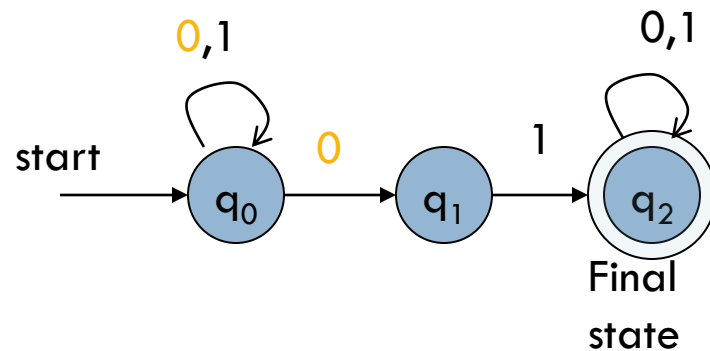
*aa*



“reject”

# NFA for strings containing 01

Why is this non-deterministic?



What will happen if at state  $q_1$  an input of 0 is received?

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{0, 1\}$
- start state =  $q_0$
- $F = \{q_2\}$
- Transition table

|          |        | symbols        |           |
|----------|--------|----------------|-----------|
| $\delta$ |        | 0              | 1         |
| states   | $q_0$  | $\{q_0, q_1\}$ | $\{q_0\}$ |
|          | $q_1$  | $\Phi$         | $\{q_2\}$ |
|          | $*q_2$ | $\{q_2\}$      | $\{q_2\}$ |

# Differences: DFA vs. NFA

## □ DFA

1. All transitions are deterministic
  - Each transition leads to exactly one state
2. For each state, transition on all possible symbols (alphabet) should be defined
3. Accepts input if the last state visited is in F
4. Sometimes harder to construct because of the number of states
5.  $\delta: Q \times \Sigma \rightarrow Q$  is a transition function

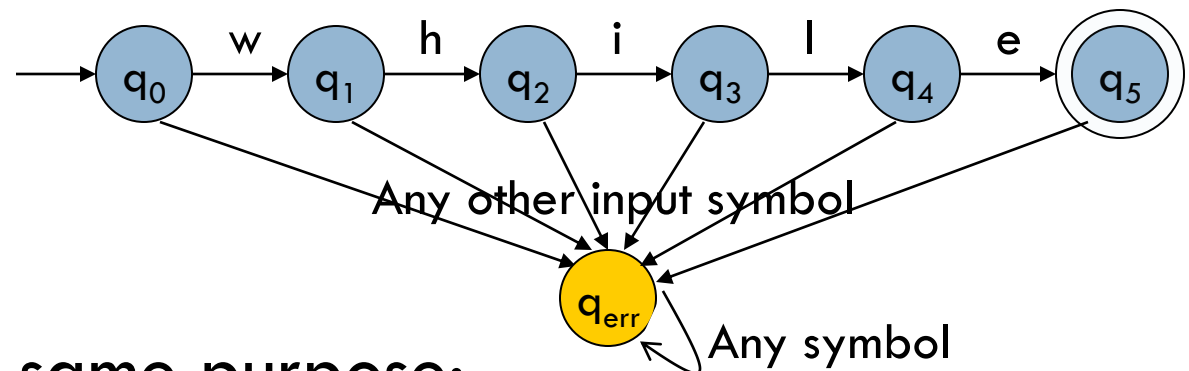
## □ NFA

1. Some transitions could be non-deterministic
  - A transition could lead to a subset of states
2. Not all symbol transitions need to be defined explicitly (if undefined will go to an error state – this is just a design convenience, not to be confused with “non-determinism”)
3. Accepts input if *one of* the last states is in F
4. Generally easier than a DFA to construct
5.  $Q \times \Sigma \rightarrow 2^Q$  [powerset of Q]

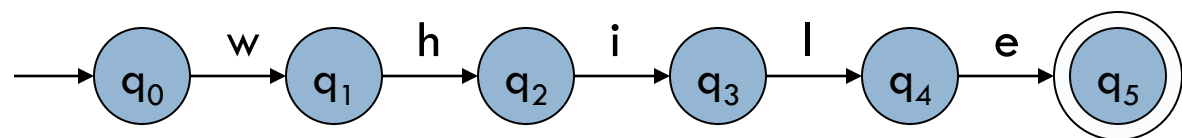


# What is an “error state” or dummy state?

- A DFA for recognizing the key word “while”



An NFA for the same purpose:

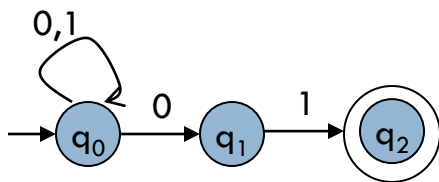


Transitions into a dead state are implicit

# NFA to DFA construction: Example

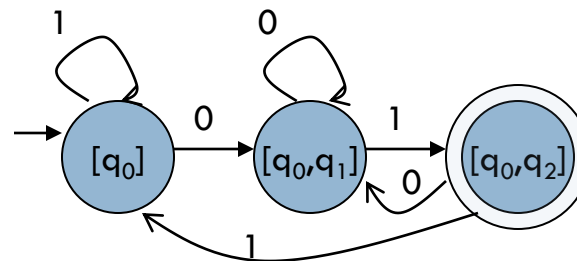
□  $L = \{w \mid w \text{ ends in } 01\}$

**NFA:**



| $\delta_N$        | 0              | 1           |
|-------------------|----------------|-------------|
| $\rightarrow q_0$ | $\{q_0, q_1\}$ | $\{q_0\}$   |
| $q_1$             | $\emptyset$    | $\{q_2\}$   |
| $*q_2$            | $\emptyset$    | $\emptyset$ |

**DFA:**



| $\delta_D$          |  |
|---------------------|--|
| $\emptyset$         |  |
| $\rightarrow [q_0]$ |  |
| $[q_1]$             |  |
| $*[q_2]$            |  |
| $[q_0, q_1]$        |  |
| $*[q_0, q_2]$       |  |
| $*[q_1, q_2]$       |  |
| $*[q_0, q_1, q_2]$  |  |

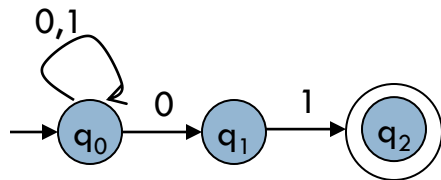
| $\delta_D$          | 0            | 1            |
|---------------------|--------------|--------------|
| $\rightarrow [q_0]$ | $[q_0, q_1]$ | $[q_0]$      |
| $[q_0, q_1]$        | $[q_0, q_1]$ | $[q_0, q_2]$ |
| $*[q_0, q_2]$       | $[q_0, q_1]$ | $[q_0]$      |

0. Enumerate all possible subsets
1. Determine transitions
2. Retain only those states reachable from  $\{q_0\}$

# NFA to DFA Contd...

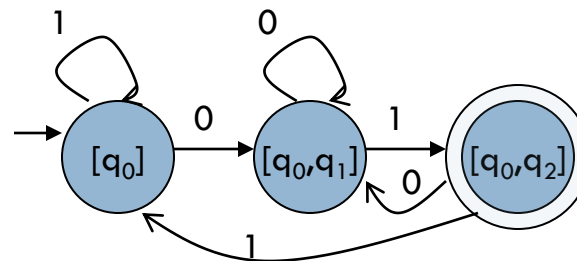
□  $L = \{w \mid w \text{ ends in } 01\}$

**NFA:**



| $\delta_N$        | 0              | 1           |
|-------------------|----------------|-------------|
| $\rightarrow q_0$ | $\{q_0, q_1\}$ | $\{q_0\}$   |
| $q_1$             | $\emptyset$    | $\{q_2\}$   |
| $*q_2$            | $\emptyset$    | $\emptyset$ |

**DFA:**



| $\delta_D$ | 0            | 1       |
|------------|--------------|---------|
| $[q_0]$    | $[q_0, q_1]$ | $[q_0]$ |

**Main Idea:**

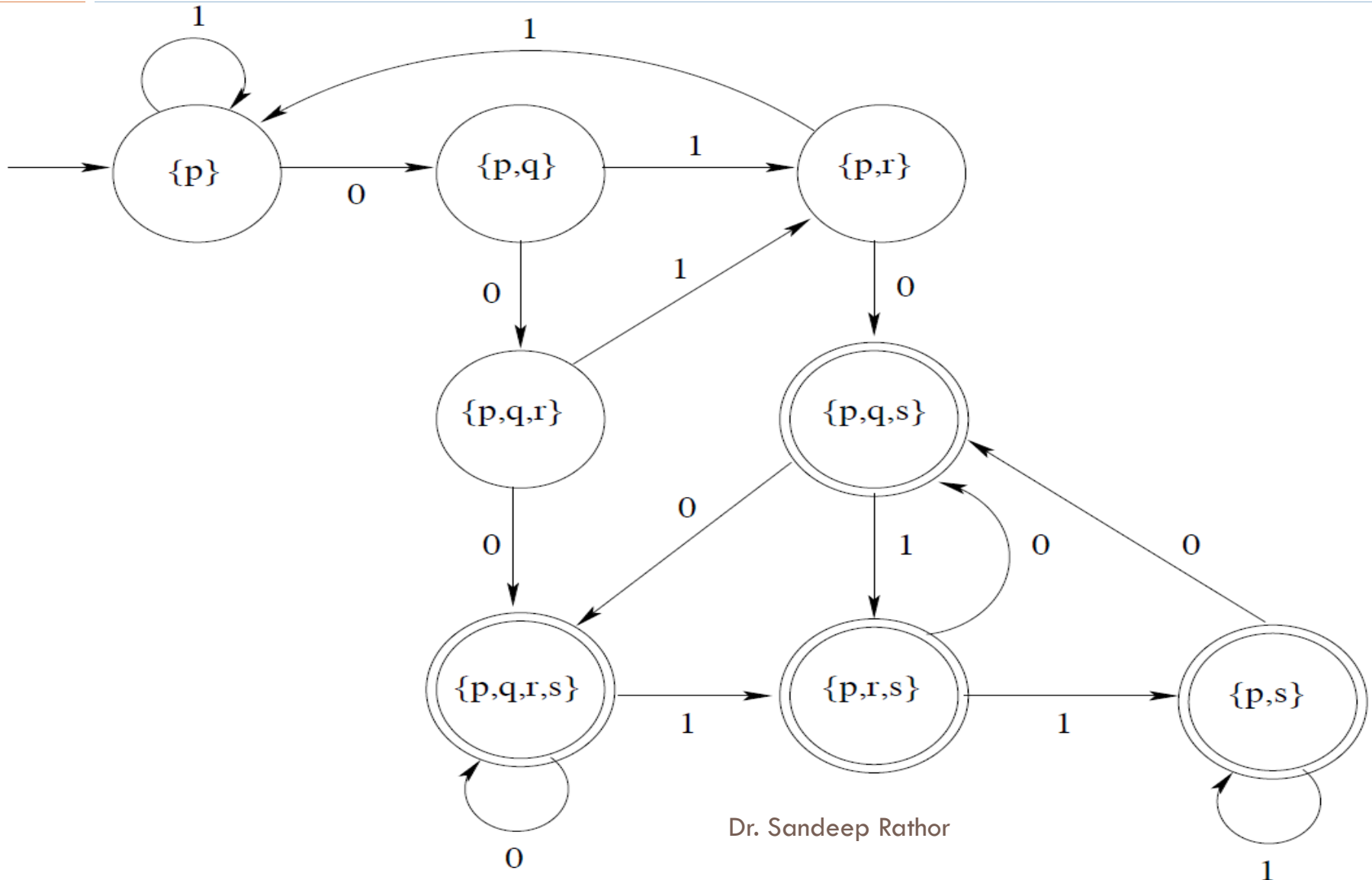
Introduce states as you go  
(on a need basis)

# Convert the given NFA to DFA

| States     | 0         | 1       |
|------------|-----------|---------|
| ->[p]      | [p,q]     | [p]     |
| [p,q]      | [p,q,r]   | [p,r]   |
| [p,r]      | [p,q,s]   | [p]     |
| [p,q,r]    | [p,q,r,s] | [p,r]   |
| [p,q,s]+   | [p,q,r,s] | [p,r,s] |
| [p,r,s]+   | [p,q,s]   | [p,s]   |
| [p,s]+     | [p,q,s]   | [p,s]   |
| [p,q,r,s]+ | [p,q,r,s] | [p,r,s] |

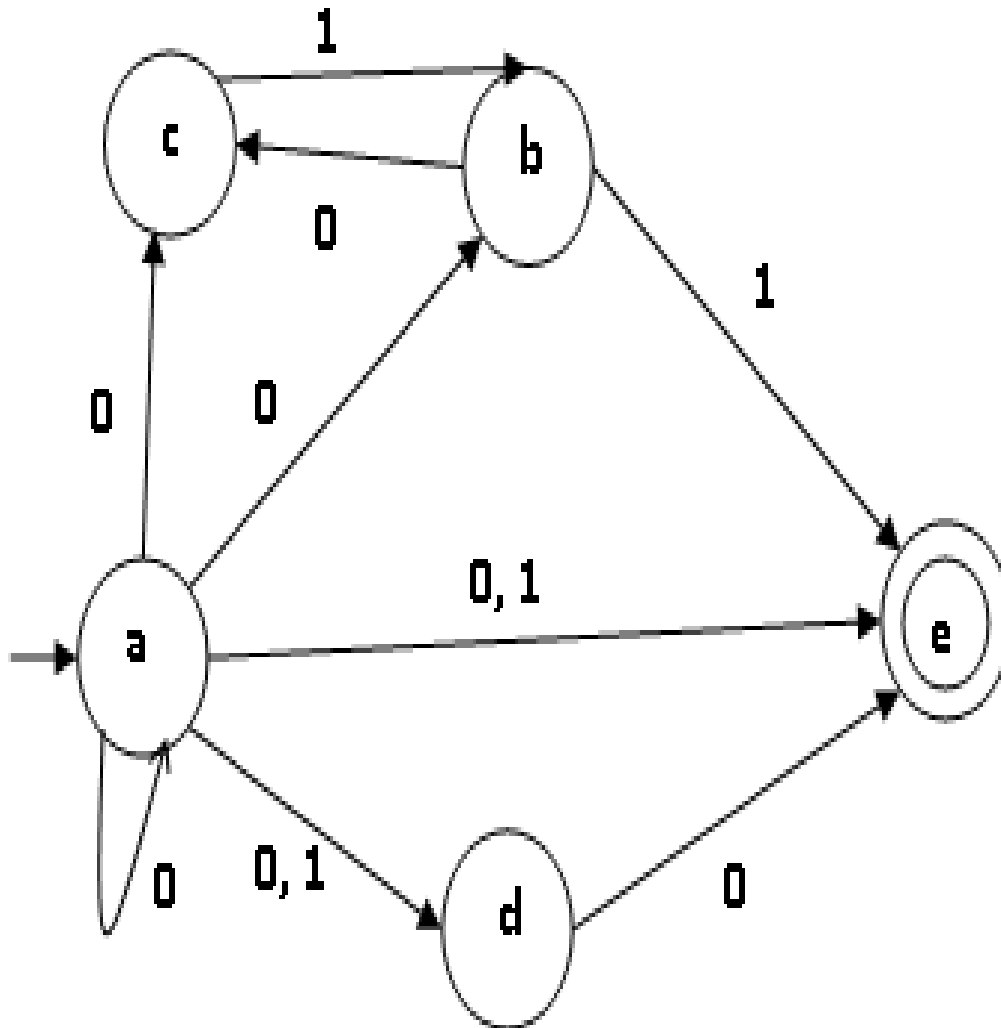
|                 | 0          | 1       |
|-----------------|------------|---------|
| $\rightarrow p$ | $\{p, q\}$ | $\{p\}$ |
| $q$             | $\{r\}$    | $\{r\}$ |
| $r$             | $\{s\}$    | $\{\}$  |
| $*s$            | $\{s\}$    | $\{s\}$ |

# Transition diagram



# Practice Session: NFA to DFA

# 1. Convert NFA to DFA



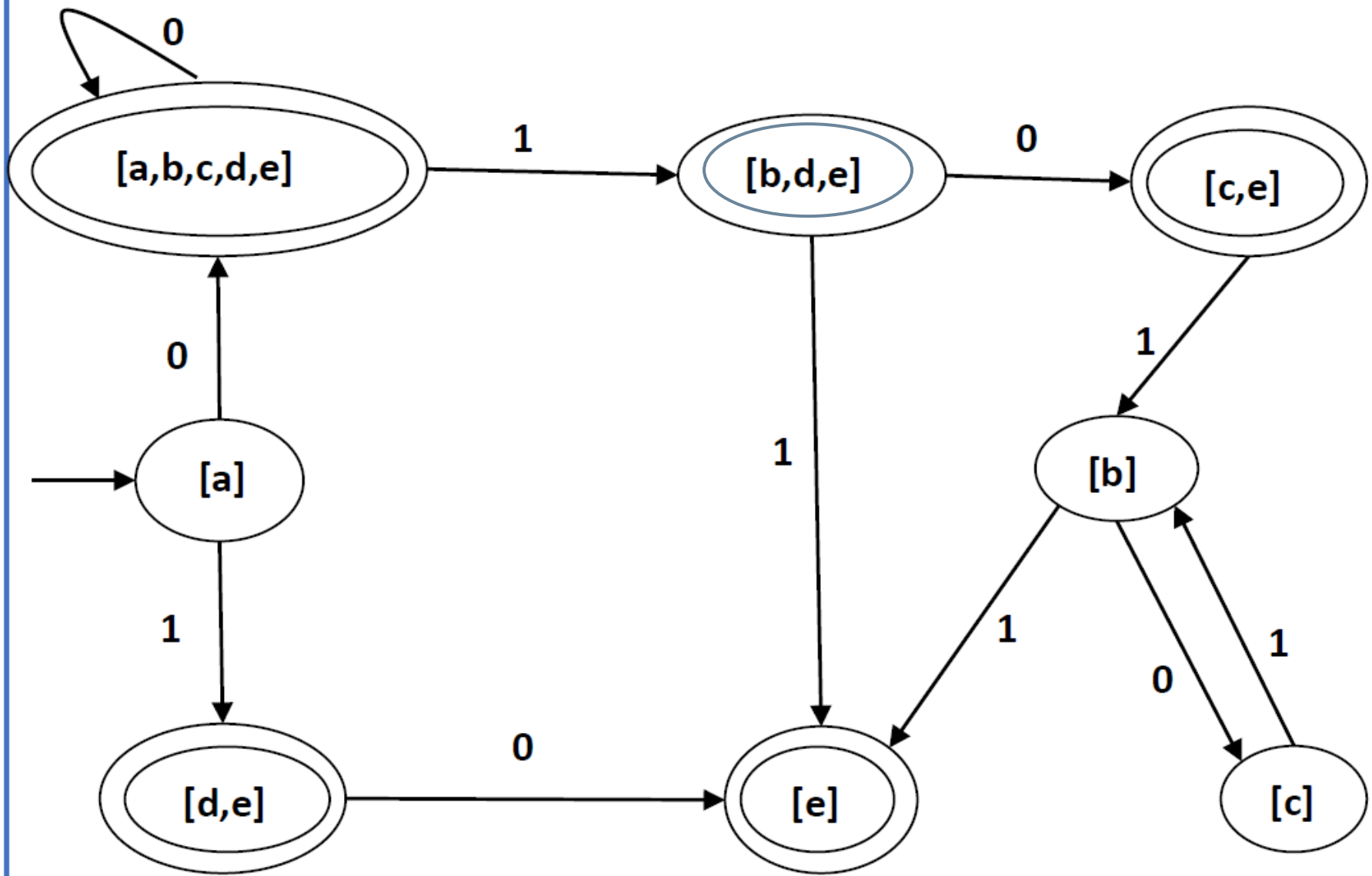
| q | $\delta(q,0)$ | $\delta(q,1)$ |
|---|---------------|---------------|
| a | {a,b,c,d,e}   | {d,e}         |
| b | {c}           | {e}           |
| c | $\emptyset$   | {b}           |
| d | {e}           | $\emptyset$   |
| e | $\emptyset$   | $\emptyset$   |

| <b>q</b>                            | <b><math>\delta(q,0)</math></b> | <b><math>\delta(q,1)</math></b> |
|-------------------------------------|---------------------------------|---------------------------------|
| <b><math>\rightarrow [a]</math></b> | <b><math>[a,b,c,d,e]</math></b> | <b><math>[d,e]</math></b>       |
| <b><math>[a,b,c,d,e]^*</math></b>   | <b><math>[a,b,c,d,e]</math></b> | <b><math>[b,d,e]</math></b>     |
| <b><math>[d,e]^*</math></b>         | <b><math>[e]</math></b>         | <b><math>\emptyset</math></b>   |
| <b><math>[b,d,e]^*</math></b>       | <b><math>[c,e]</math></b>       | <b><math>[e]</math></b>         |
| <b><math>[e]^*</math></b>           | <b><math>\emptyset</math></b>   | <b><math>\emptyset</math></b>   |
| <b><math>[c,e]^*</math></b>         | <b><math>\emptyset</math></b>   | <b><math>[b]</math></b>         |
| <b><math>[b]</math></b>             | <b><math>[c]</math></b>         | <b><math>[e]</math></b>         |
| <b><math>[c]</math></b>             | <b><math>\emptyset</math></b>   | <b><math>[b]</math></b>         |

Required DFA

Dr. Sandeep Rathor  
Transition diagram on next page...





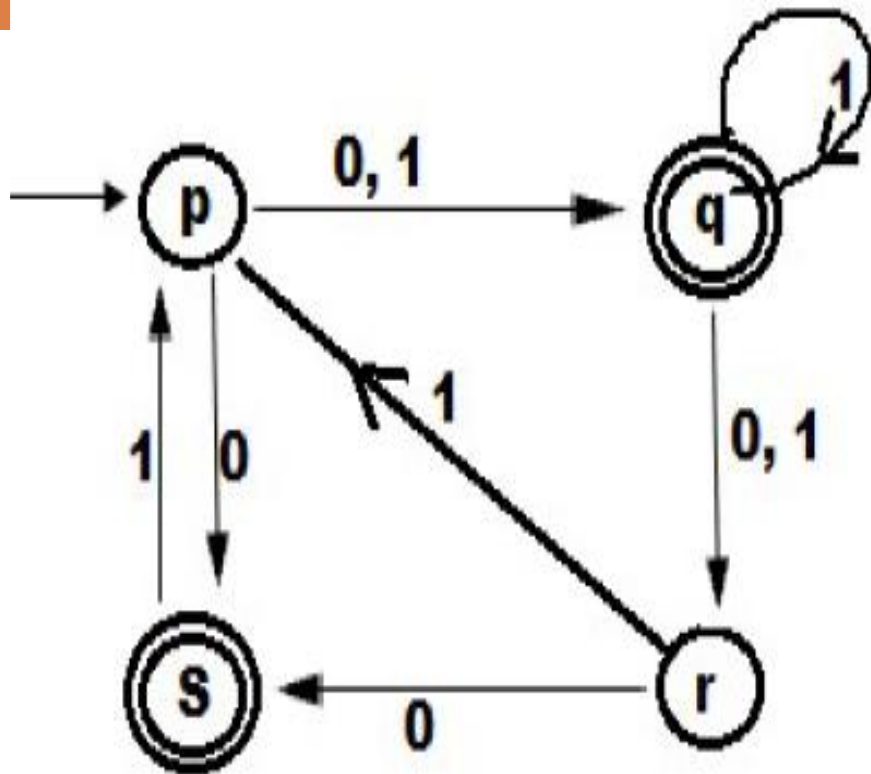
## 2. Convert into DFA

| States | 0 | 1   |
|--------|---|-----|
| ->p+   | p | q   |
| q      | q | p,q |

DFA equivalent to given NFA

| States  | 0     | 1     |
|---------|-------|-------|
| ->[p]+  | [p]   | [q]   |
| [q]     | [q]   | [p,q] |
| [p,q] + | [p,q] | [p,q] |

# 3. Convert given NFA to DFA



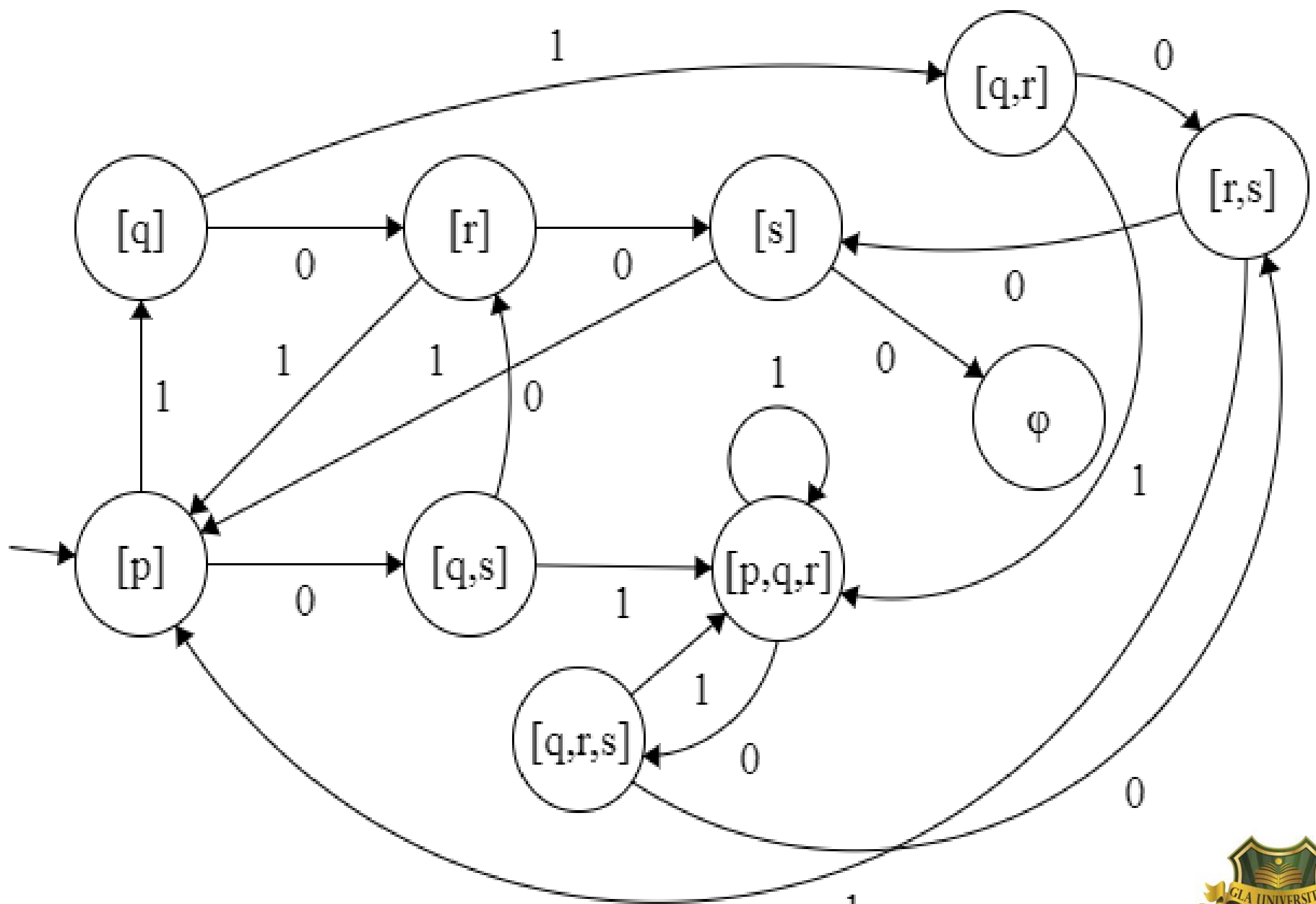
|     | 0    | 1    |
|-----|------|------|
| → p | q, s | q    |
| *q  | r    | q, r |
| r   | s    | p    |
| *s  | -    | p    |

# Required DFA

| State/ Input | 0         | 1         |
|--------------|-----------|-----------|
| →[p]         | [q, s]    | [q]       |
| *[q]         | [r]       | [q, r]    |
| [r]          | [s]       | [p]       |
| *[s]         | $\varphi$ | [p]       |
| *[q, r]      | [r, s]    | [p, q, r] |
| *[q, s]      | [r]       | [p, q, r] |
| *[r, s]      | [s]       | [p]       |
| *[p, q, r]   | [q, r, s] | [p, q, r] |
| *[q, r, s]   | [r, s]    | [p, q, r] |

Transition Diagram...

Dr. Sandeep Rathor



Dr. Sandeep Rathor


# Practice Contd...



## 4. Convert given NFA to DFA

| States/input | 0    | 1   |
|--------------|------|-----|
| ->P+ (Final) | Q, S | Q   |
| Q+ (Final)   | R    | R,Q |
| R            | S    | S   |
| S            | -    | P   |

# Answer: Required DFA

| State/ Input   | 0       | 1         |
|--|---------|-----------|
|  [P]+ | [Q,S]   | [Q]       |
| [Q]+   | [R]     | [R,Q]     |
| [R]  | [S]     | [S]       |
| [S]  | -       | [P]       |
| [Q,S] +  | [R]     | [P,Q,R]   |
| [R,Q] +  | [R,S]   | [Q,R,S]   |
| [R,S]  | [S]     | [P,S]     |
| [P,S]+   | [Q,S]   | [P,Q]     |
| [P,Q] +  | [Q,R,S] | [R,Q]     |
| [P,Q,R] +  | [Q,R,S] | [Q,R,S]   |
| [Q,R,S] +  | [R,S]   | [P,Q,R,S] |
| [P,Q,R,S] +  | [Q,R,S] | [P,Q,R,S] |

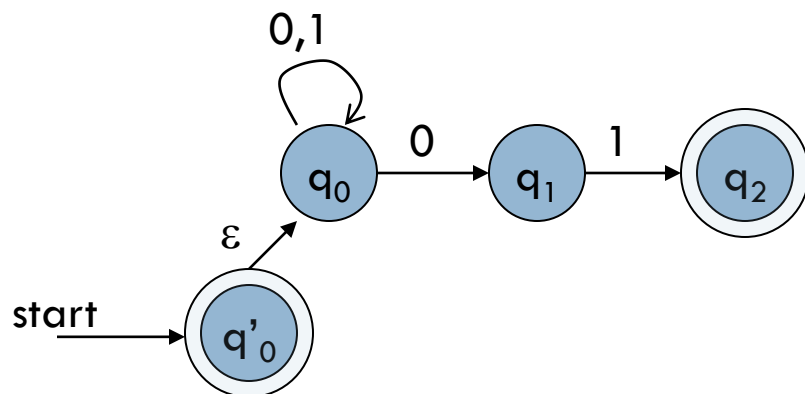
# FA with $\epsilon$ -Transitions

- $\epsilon$ -NFAs are those NFAs with at least one explicit  $\epsilon$ -transition defined.
- Explicit  $\epsilon$ -transitions is transition from one state to another state without consuming any additional input symbol



# Example of an $\epsilon$ -NFA

$L = \{w \mid w \text{ is empty, or if non-empty will end in } 01\}$



- $\epsilon$ -closure of a state  $q$ , **ECLOSE( $q$ )**, is the set of all states (including itself) that can be reached from  $q$  by repeatedly making an arbitrary number of  $\epsilon$ -transitions.

| $\delta_E$          | 0              | 1           | $\epsilon$      |
|---------------------|----------------|-------------|-----------------|
| $\rightarrow *q'_0$ | $\emptyset$    | $\emptyset$ | $\{q'_0, q_0\}$ |
| $q_0$               | $\{q_0, q_1\}$ | $\{q_0\}$   | $\{q_0\}$       |
| $q_1$               | $\emptyset$    | $\{q_2\}$   | $\{q_1\}$       |
| $*q_2$              | $\emptyset$    | $\emptyset$ | $\{q_2\}$       |

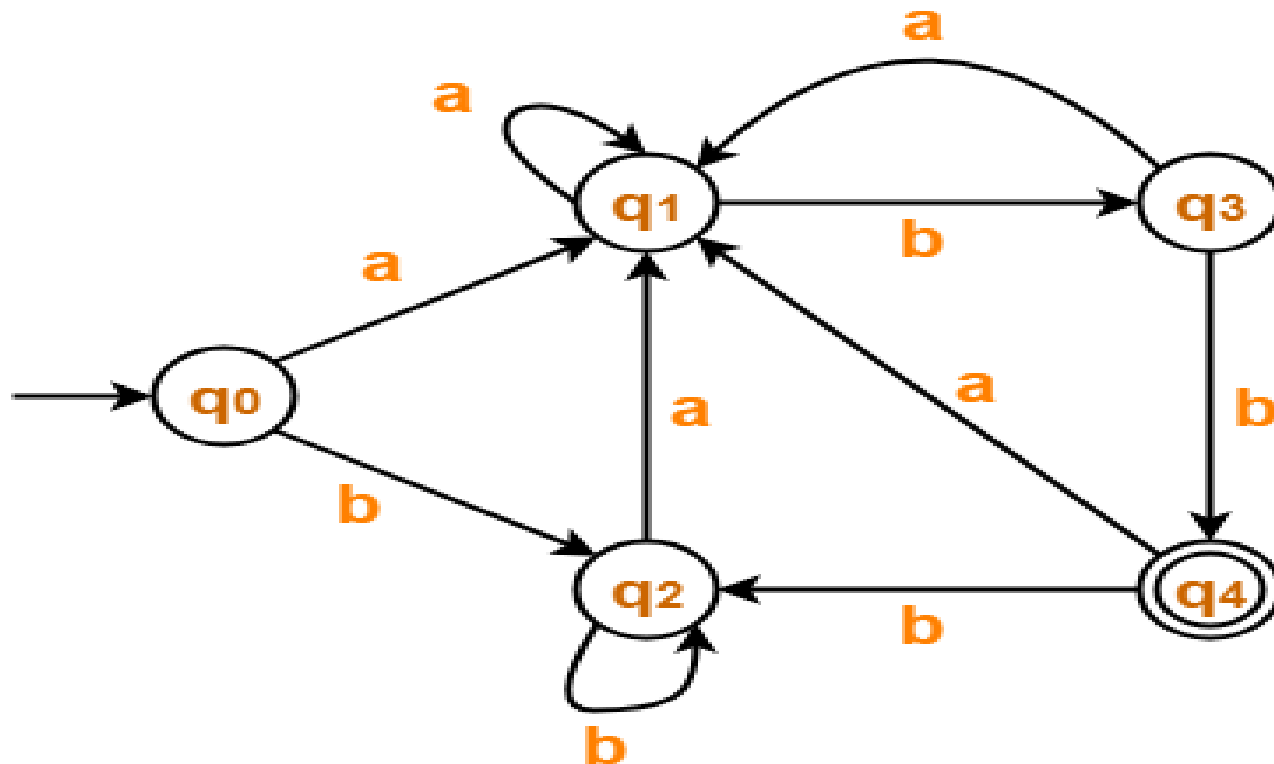
# Minimization of DFA

## Minimization of DFA (Methods)

Equivalence Theorem

Myhill Nerode Theorem  
(Table Filling Method)

# Minimized the given DFA...



# Transition table as per given diagram



|     | a  | b   |
|-----|----|-----|
| →q0 | q1 | q2  |
| q1  | q1 | q3  |
| q2  | q1 | q2  |
| q3  | q1 | *q4 |
| *q4 | q1 | q2  |

Now using Equivalence Theorem, we have-

$$\pi_0 = \{ q_0, q_1, q_2, q_3 \} \{ q_4 \}$$

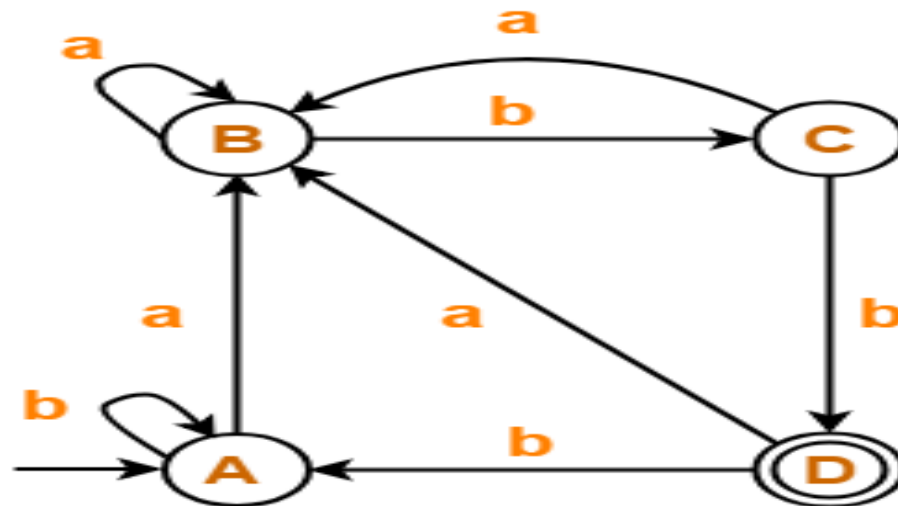
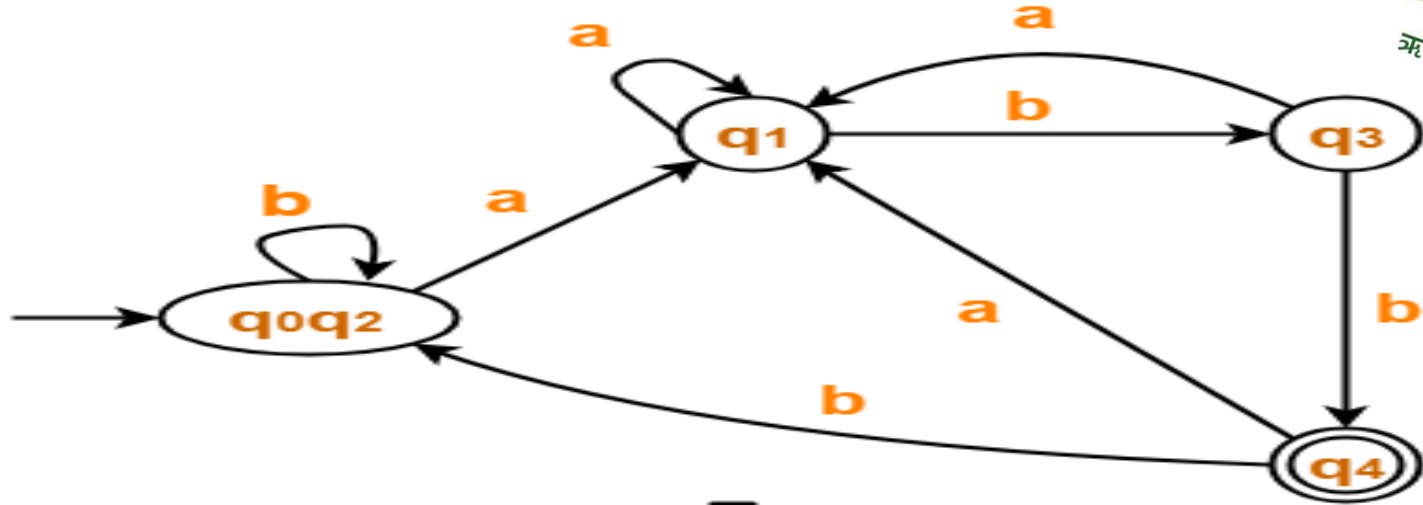
$$\pi_1 = \{ q_0, q_1, q_2 \} \{ q_3 \} \{ q_4 \}$$

$$\pi_2 = \{ q_0, q_2 \} \{ q_1 \} \{ q_3 \} \{ q_4 \}$$

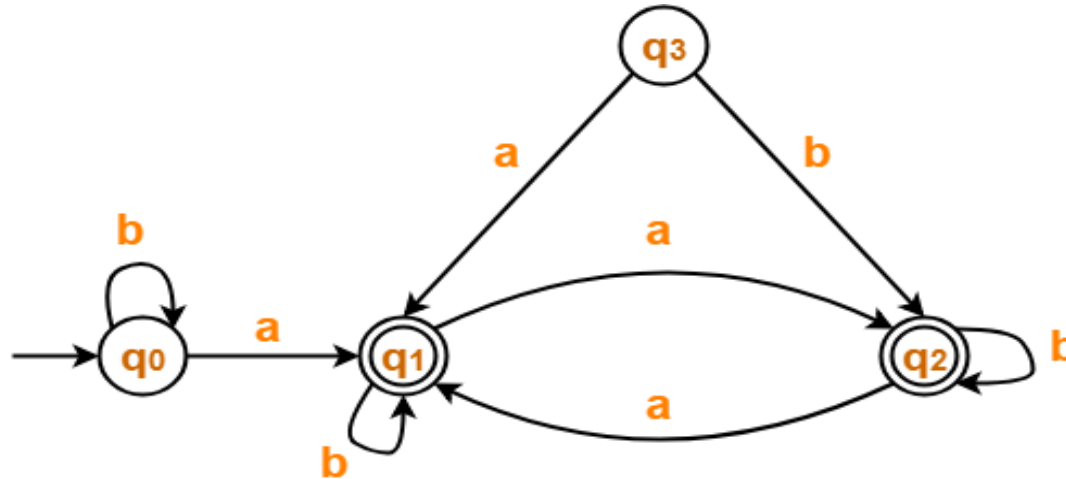
$$\pi_3 = \{ q_0, q_2 \} \{ q_1 \} \{ q_3 \} \{ q_4 \}$$

Since  $\pi_3 = \pi_2$ , so we stop.

# Minimized DFA



# Ex-2, Minimization contd...



Transition Table

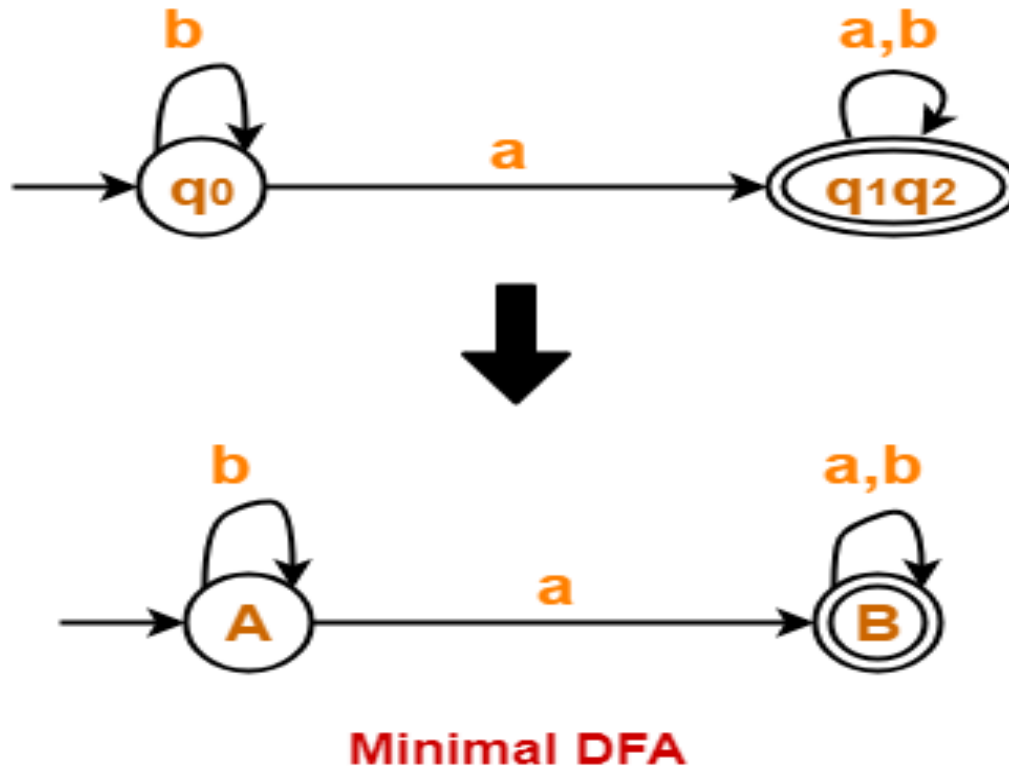
|     | a   | b   |
|-----|-----|-----|
| →q0 | *q1 | q0  |
| *q1 | *q2 | *q1 |
| *q2 | *q1 | *q2 |

$$P_0 = \{ q_0 \} \{ q_1, q_2 \}$$

$$P_1 = \{ q_0 \} \{ q_1, q_2 \}$$

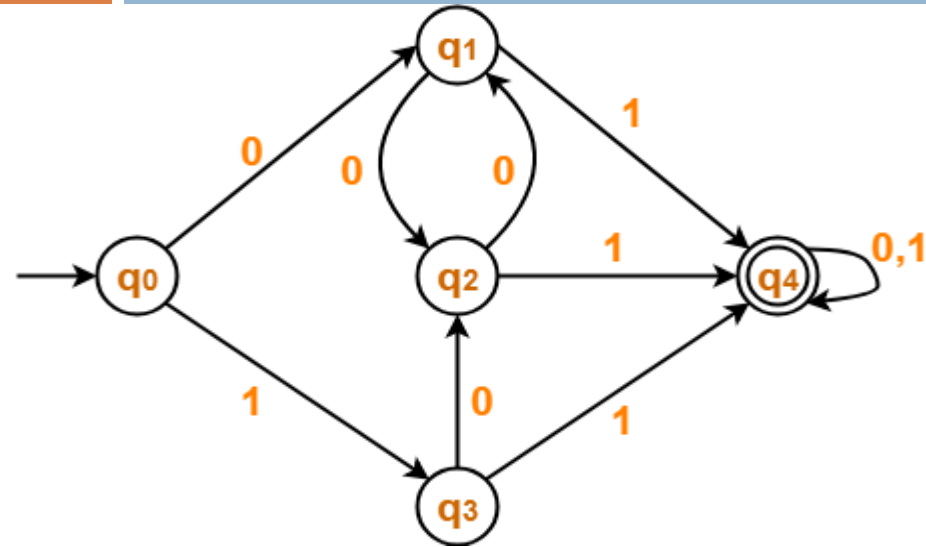
Since  $P_1 = P_0$ , so we stop.

# Minimized automata...



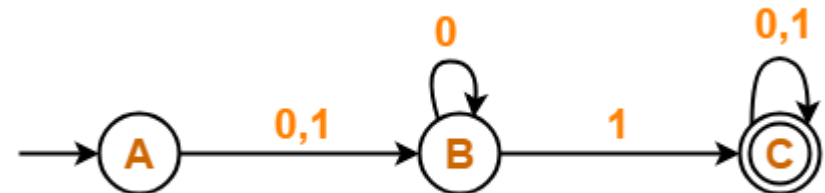
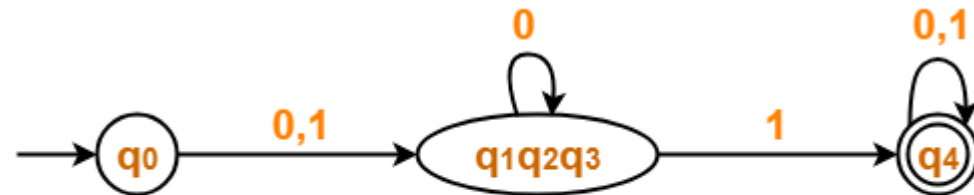
# Minimized the following:

## Example-3:



$$P_0 = \{ q_0, q_1, q_2, q_3 \} \{ q_4 \}$$
$$P_1 = \{ q_0 \} \{ q_1, q_2, q_3 \} \{ q_4 \}$$
$$P_2 = \{ q_0 \} \{ q_1, q_2, q_3 \} \{ q_4 \}$$

Minimized





## Example-4:

State  $q_5$  is inaccessible from the initial state.  
So, we eliminate it and its associated edges from the DFA.

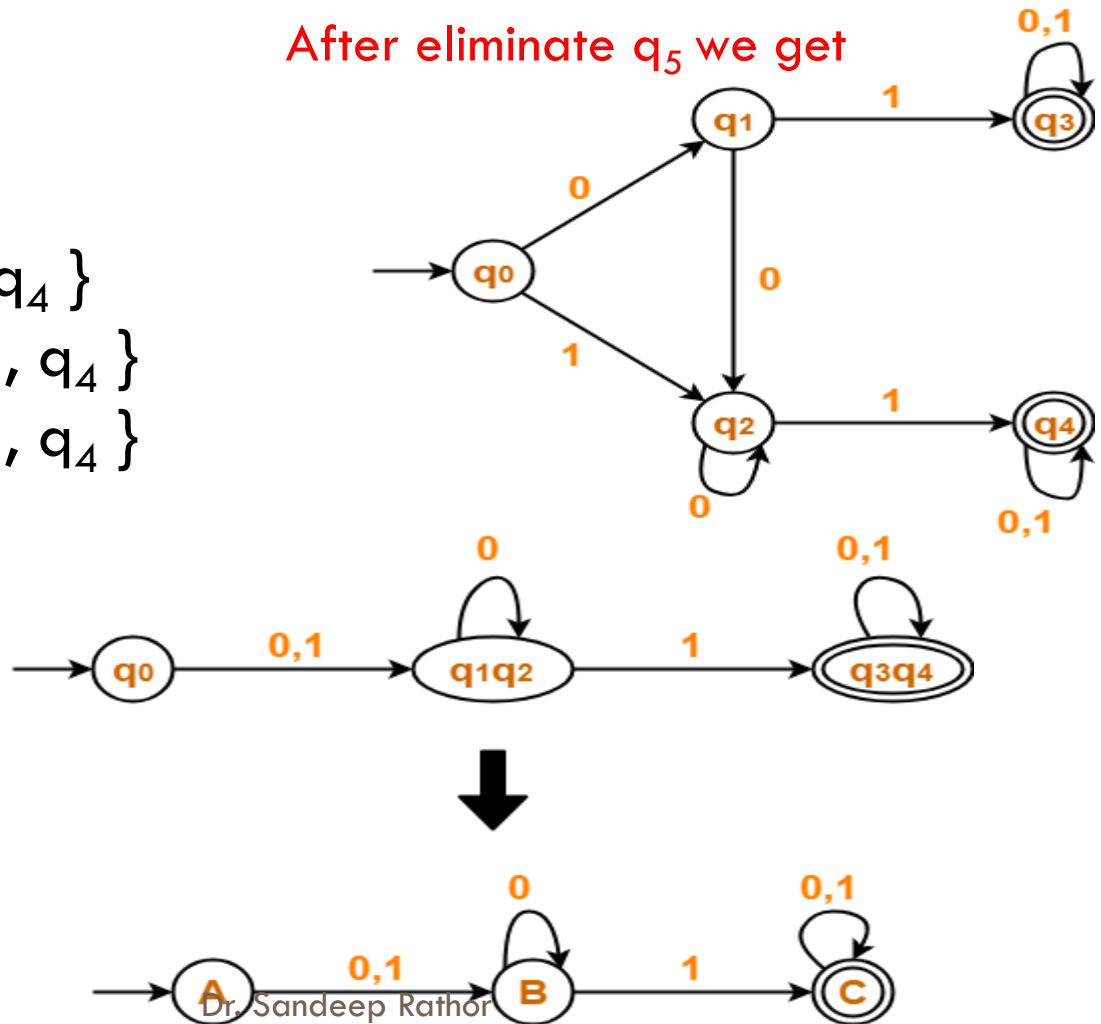
After eliminate  $q_5$  we get

$$P_0 = \{q_0, q_1, q_2\} \{q_3, q_4\}$$

$$P_1 = \{q_0\} \{q_1, q_2\} \{q_3, q_4\}$$

$$P_2 = \{q_0\} \{q_1, q_2\} \{q_3, q_4\}$$

Since  $P_2 = P_1$ , so we st



Minimal DFA

# Example-5 for Practice...

Minimize the given Automata

| States/input     | 0  | 1  |
|------------------|----|----|
| ->q <sub>0</sub> | q1 | q5 |
| q1               | q6 | q2 |
| q2+ (Final)      | q0 | q2 |
| q3               | q2 | q6 |
| q4               | q7 | q5 |
| q5               | q2 | q6 |
| q6               | q6 | q4 |
| q7               | q6 | q2 |

# Minimizing (Using Equivalence theorem)

$$\pi_0 = \{\{q_2\}, \{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}\}$$

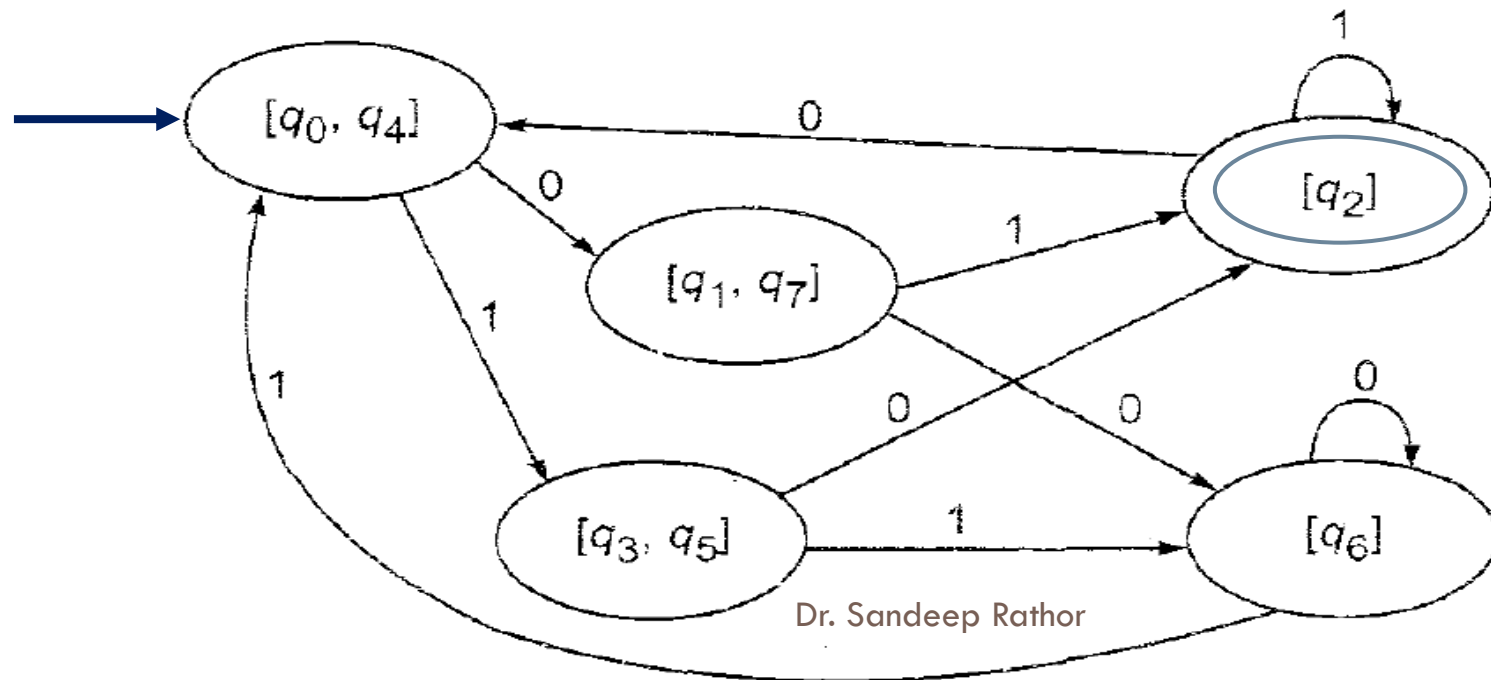
$$\pi_1 = \{\{q_2\}, \{q_0, q_4, q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$$

$$\pi_2 = \{\{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$$

$$\pi_3 = \{\{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$$

$$\pi_2 = \pi_3,$$

| State/ $\Sigma$          | 0            | 1            |
|--------------------------|--------------|--------------|
| $\rightarrow [q_0, q_4]$ | $[q_1, q_7]$ | $[q_3, q_5]$ |
| $[q_1, q_7]$             | $[q_6]$      | $[q_2]$      |
| $[q_2]$                  | $[q_0, q_4]$ | $[q_2]$      |
| $[q_3, q_5]$             | $[q_2]$      | $[q_6]$      |
| $[q_6]$                  | $[q_6]$      | $[q_0, q_4]$ |



# Example-6 for Practice...

Minimize the given Automata

| States/input             | a              | b              |
|--------------------------|----------------|----------------|
| ->q <sub>0</sub>         | q <sub>1</sub> | q <sub>0</sub> |
| q <sub>1</sub>           | q <sub>0</sub> | q <sub>2</sub> |
| q <sub>2</sub>           | q <sub>3</sub> | q <sub>1</sub> |
| q <sub>3</sub> + (Final) | q <sub>3</sub> | q <sub>0</sub> |
| q <sub>4</sub>           | q <sub>3</sub> | q <sub>5</sub> |
| q <sub>5</sub>           | q <sub>6</sub> | q <sub>4</sub> |
| q <sub>6</sub>           | q <sub>5</sub> | q <sub>6</sub> |

# Minimizing (Using Equivalence theorem)

$$\pi_0 = \{\{q_3\}, \{q_0, q_1, q_2, q_4, q_5, q_6\}$$

$$\pi_1 = \{\{q_3\}, \{q_0, q_1, q_5, q_6\}, \{q_2, q_4\}$$

$$\pi_2 = \{\{q_3\}, \{q_0, q_6\}, \{q_1, q_5\}, \{q_2, q_4\}$$

$$\pi_3 = \{\{q_3\}, \{q_0, q_6\}, \{q_1, q_5\}, \{q_2, q_4\}$$

$$\pi_3 = \pi_2$$

State/ $\Sigma$

$a$

$b$

$[q_0, q_6]$

$[q_1, q_5]$

$[q_0, q_6]$

$[q_1, q_5]$

$[q_0, q_6]$

$[q_2, q_4]$

$[q_2, q_4]$

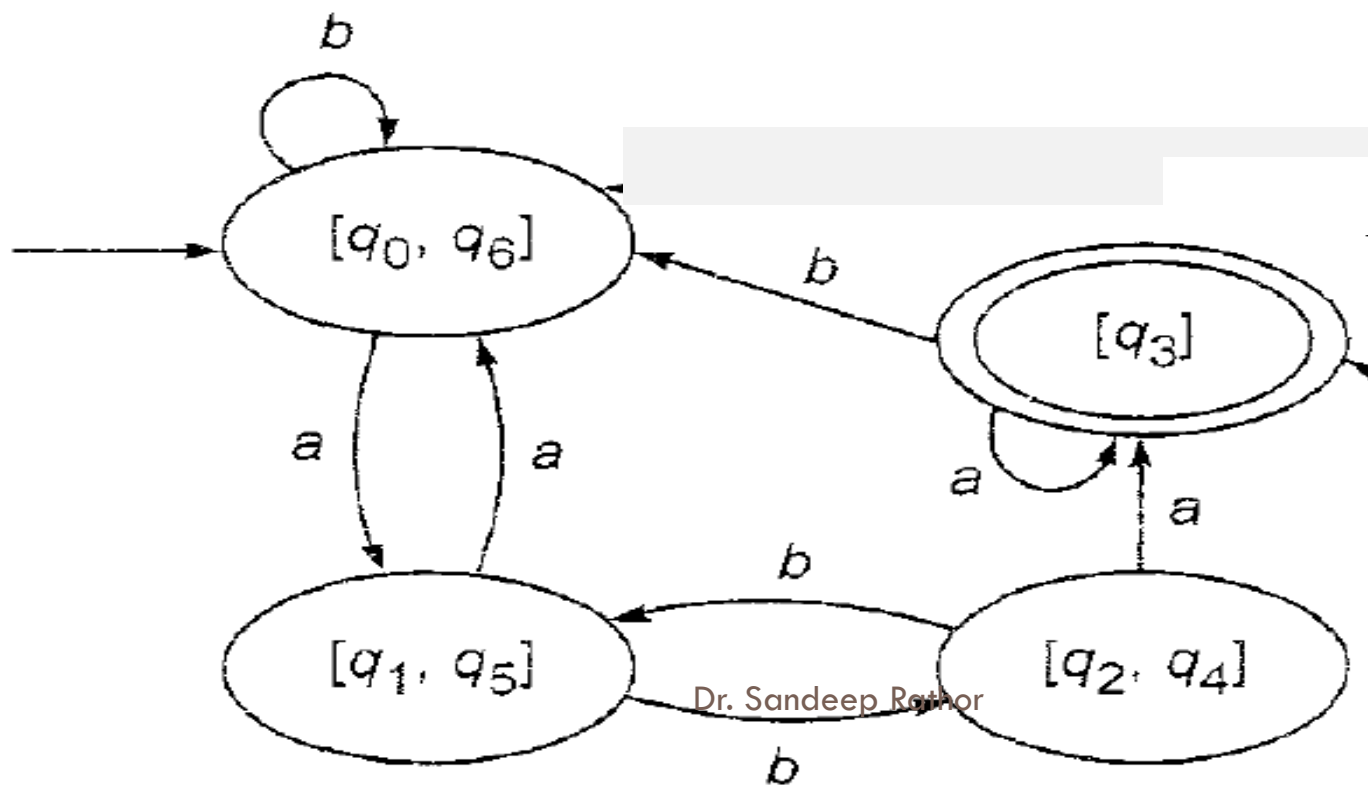
$[q_3]$

$[q_1, q_5]$

$[q_3]$

$[q_3]$

$[q_0, q_6]$



# DFA Minimization using Myhill-Nerode Theorem

## *Algorithm*

**Input** – DFA

**Output** – Minimized DFA

**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**]

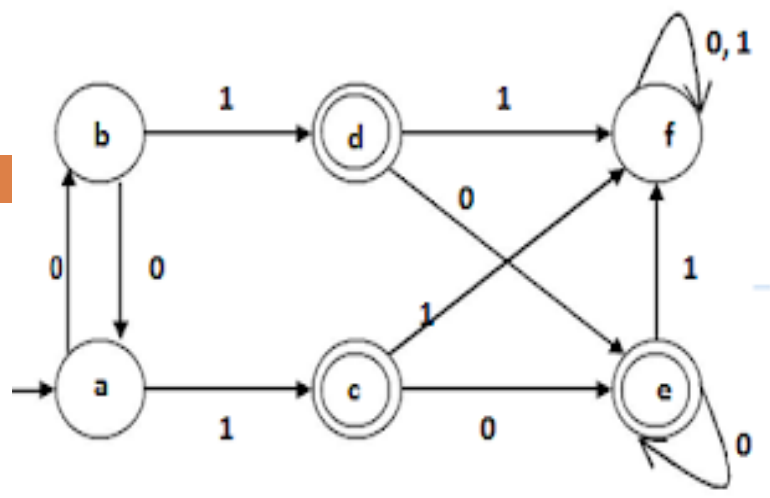
**Step 3** – Repeat this step until we cannot mark anymore states –  
If there is an unmarked pair  $(Q_i, Q_j)$ , mark it if the pair  $\{\delta(Q_i, A), \delta(Q_j, A)\}$  is marked for some input alphabet.

**Step 4** – Combine all the unmarked pair  $(Q_i, Q_j)$  and make them a single state in the reduced DFA.



Step 2 – We mark the state pairs

# Minimization using Myhill-Nerode Theorem



State Diagram of DFA

Step 1 : We draw a table for all pair of states.

|   | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| a |   |   |   |   |   |   |
| b |   |   |   |   |   |   |
| c |   |   |   |   |   |   |
| d |   |   |   |   |   |   |
| e |   |   |   |   |   |   |
| f |   |   |   |   |   |   |

Step 2 : We mark the state pairs.

|   | a                  | b | c | d | e | f |
|---|--------------------|---|---|---|---|---|
| a |                    |   |   |   |   |   |
| b |                    |   |   |   |   |   |
| c | ✓                  | ✓ |   |   |   |   |
| d | ✓                  | ✓ |   |   |   |   |
| e | ✓                  | ✓ |   |   |   |   |
| f | Dr. Sandeep Rathor |   | ✓ | ✓ | ✓ |   |

**Step 3** – We will try to mark the state pairs, with green colored check mark, transitively. If we input 1 to state 'a' and 'f', it will go to state 'c' and 'f' respectively. (c, f) is already marked, hence we will mark pair (a, f). Now, we input 1 to state 'b' and 'f'; it will go to state 'd' and 'f' respectively. (d, f) is already marked, hence we will mark pair (b, f).

|   | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| a |   |   |   |   |   |   |
| b |   |   |   |   |   |   |
| c | ✓ | ✓ |   |   |   |   |
| d | ✓ | ✓ |   |   |   |   |
| e | ✓ | ✓ |   |   |   |   |
| f | ✓ | ✓ | ✓ | ✓ | ✓ |   |

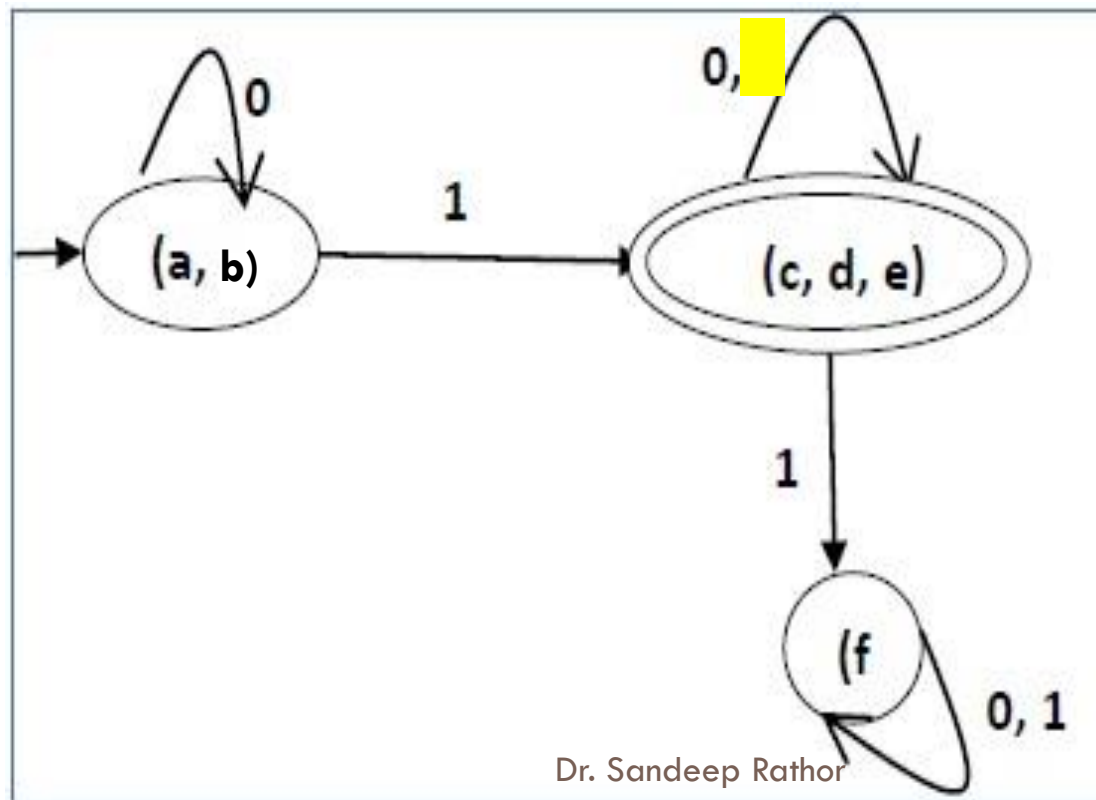
After step 3, we have got state combinations {a, b} {c, d} {c, e} {d, e} that are unmarked.

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

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

# Minimization using Myhill-Nerode Contd...

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



# Finite Automata with Output

DFA, NFA, s -NFA are FA without outputs (language acceptors)

**Language transducers:** Produces output on input

Finite automata may have outputs corresponding to each transition.

There are two types of finite state machines that generate output –

- **Moore Machine**

- **Mealy Machine**

# Moore Machine

A Moore machine can be described by a 6 tuple

i.e.  $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$  where –

$Q$  is a finite set of states.

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

$\Delta$  is a finite set of symbols called the output alphabet.

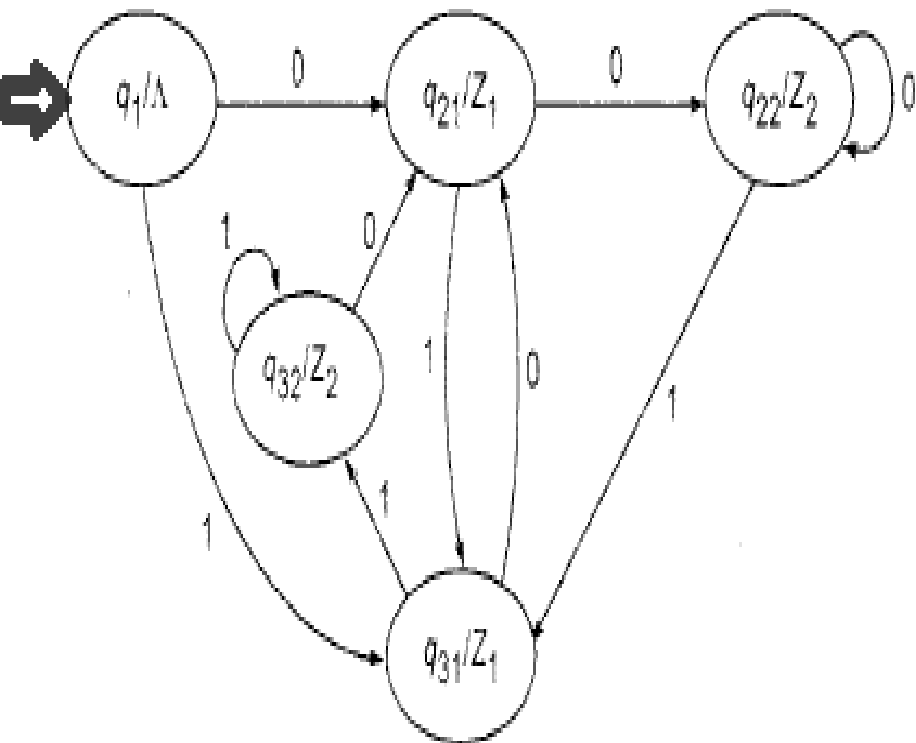
$\delta$  is the input transition function where  $\delta: Q \times \Sigma \rightarrow Q$

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

$q_0$  is the initial state ( $q_0 \in Q$ ).

# Example of Moore Machine

| Present state | Next State |           | Output |
|---------------|------------|-----------|--------|
|               | Input = 0  | Input = 1 |        |
| → a           | b          | c         | x2     |
| b             | b          | d         | x1     |
| c             | c          | d         | x2     |
| d             | d          | d         | x3     |

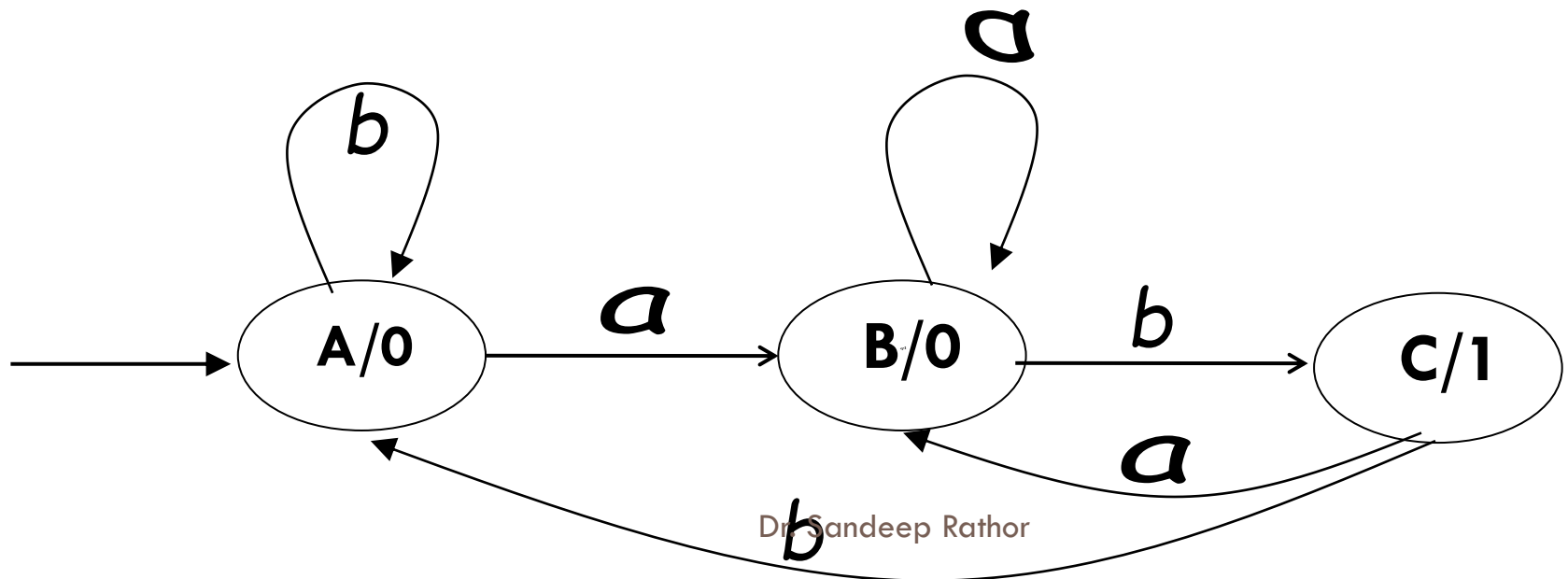


| Present State     | Next State |     | Output    |
|-------------------|------------|-----|-----------|
|                   | 0          | 1   | $\Delta$  |
| $\rightarrow q_0$ | q21        | q31 | $\Lambda$ |
| q21               | q22        | q31 | Z1        |
| q22               | q22        | q31 | Z2        |
| q31               | q21        | q32 | Z1        |
| q32               | q21        | q32 | Z2        |

# Construction of Moore Machine

**Question1:** Construct a Moore machine that takes set of all strings  $\{a,b\}$  as input and prints '1' as output for every occurrence of 'ab' as substring.

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

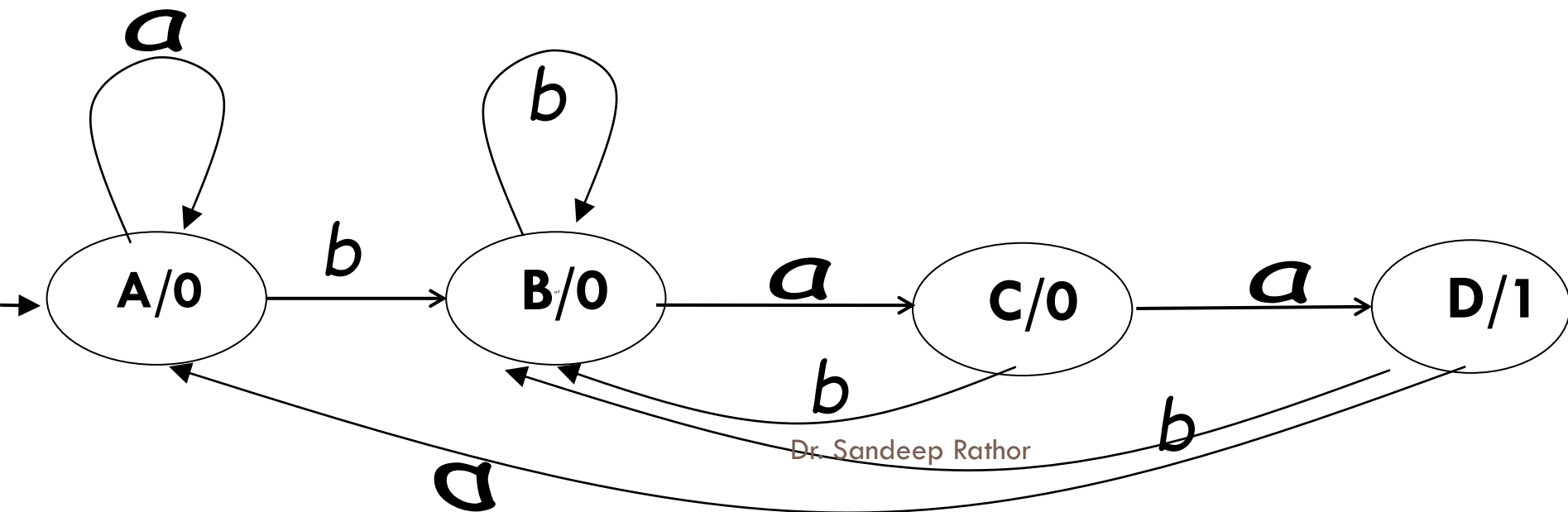




# Construction of Moore Machine Contd...

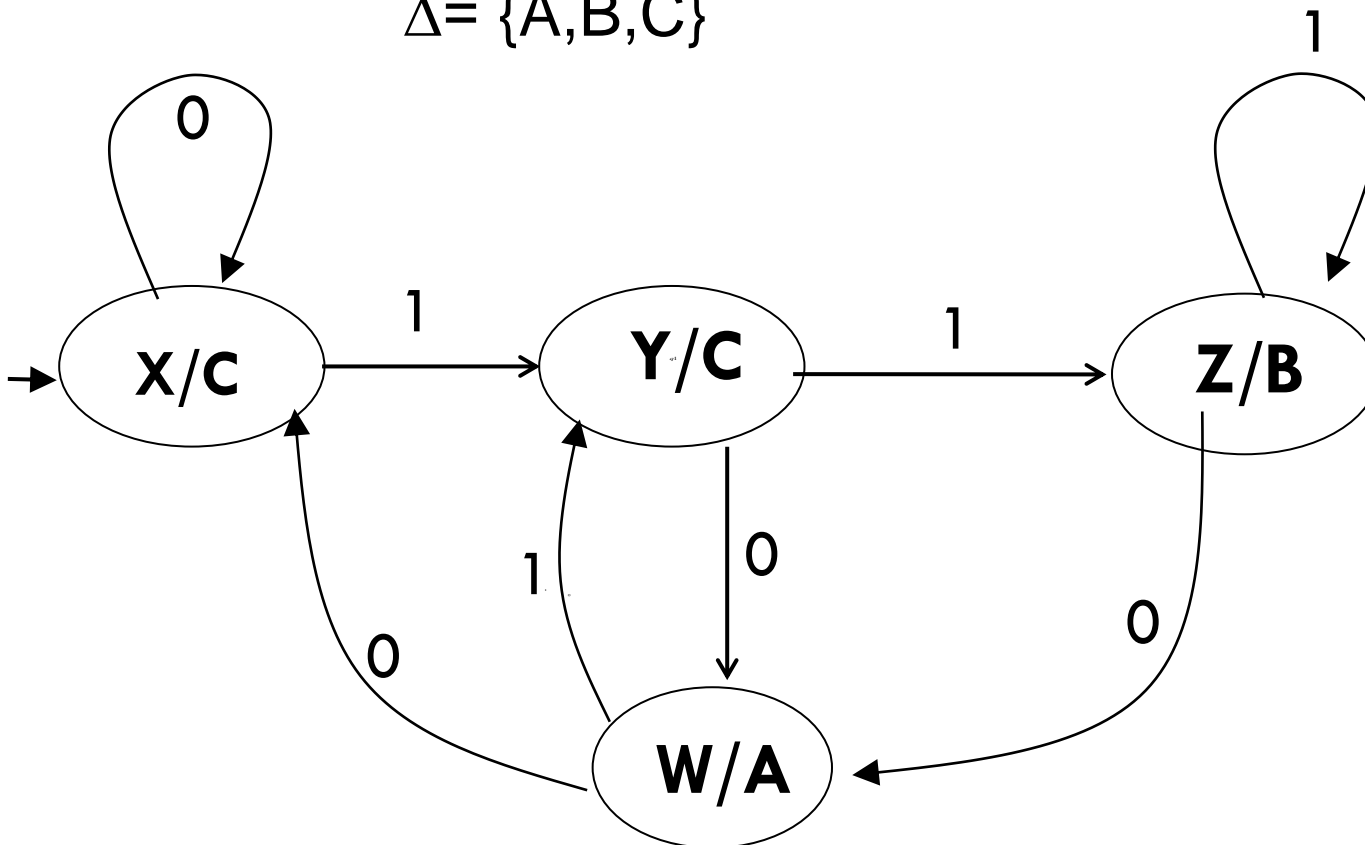
**Question2:** Construct a Moore machine that takes set of all strings over  $\{a,b\}$  and counts no. of occurrences of substring 'baa'.

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



**Question3:** Construct a Moore machine that takes set of all strings over  $\{0,1\}$  and produce 'A' as output if input ends with '10' or produces 'B' as output if ends with '11' otherwise 'C'.

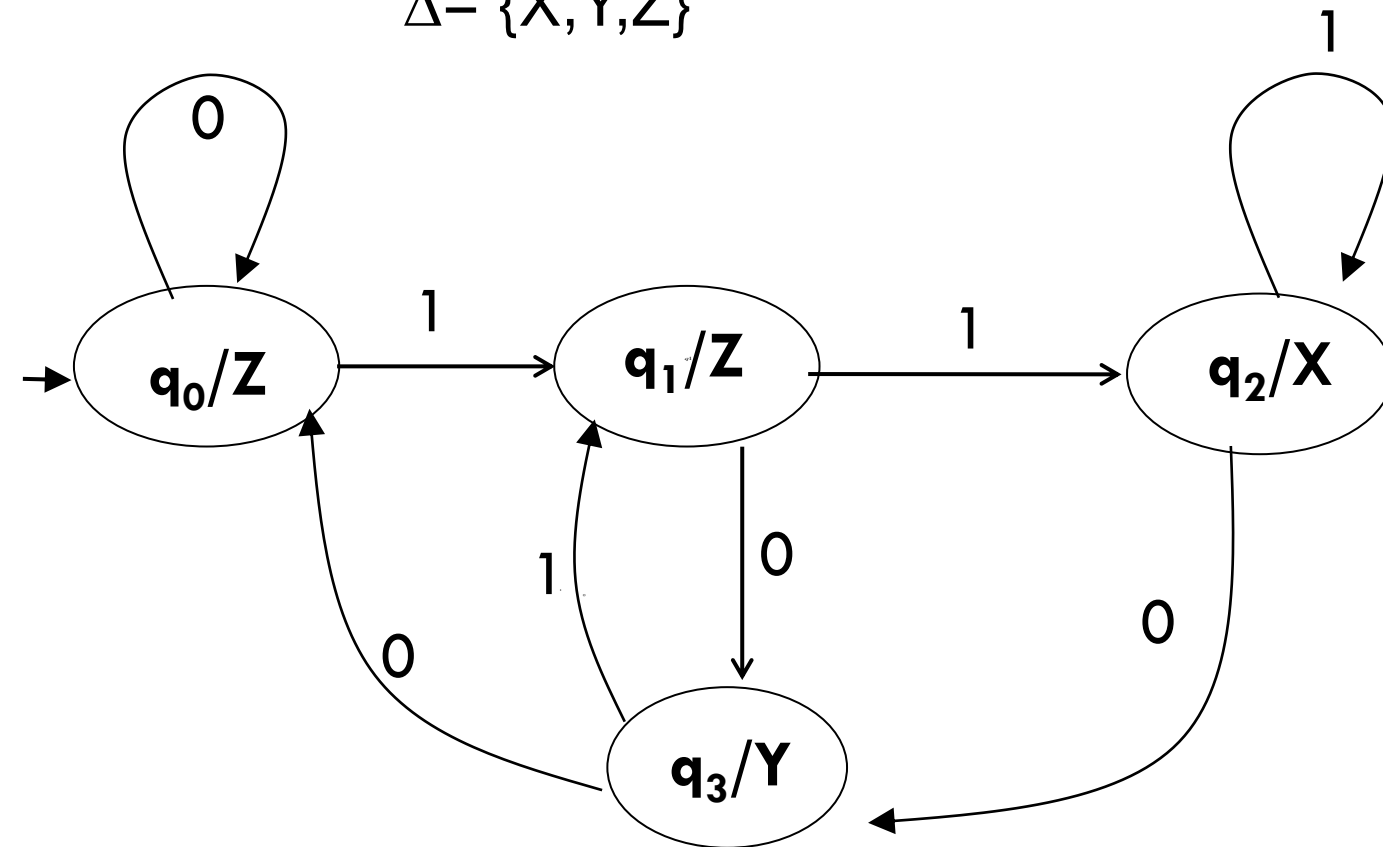
Solution:  $\Sigma = \{0,1\}$   
 $\Delta = \{A,B,C\}$



**Question4:** Construct a Moore machine that takes set of all strings over  $\{0,1\}$  and produce 'X' as output if input ends with '11' or produces 'Y' as output if ends with '10' otherwise 'Z'.

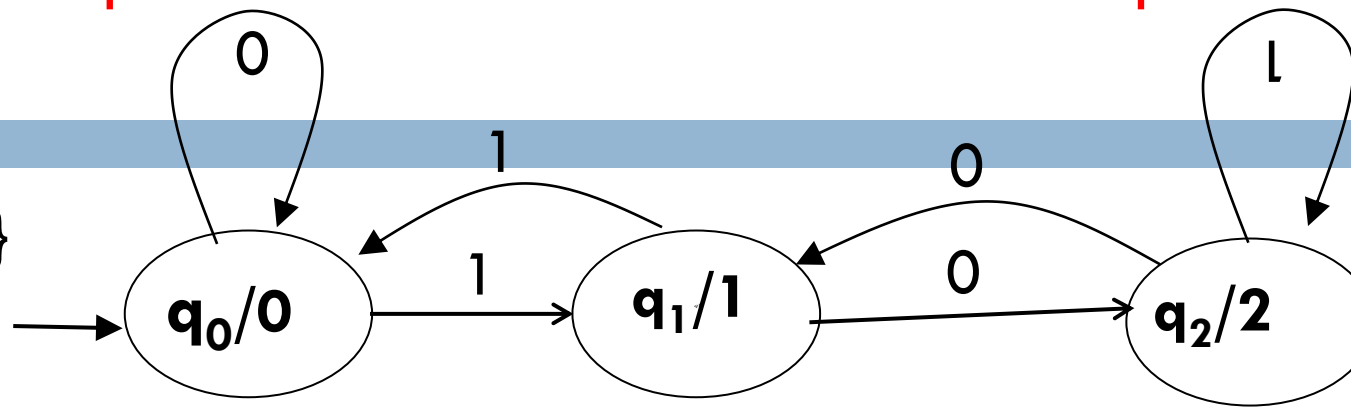
Solution:

$$\Sigma = \{0,1\}$$
$$\Delta = \{X,Y,Z\}$$



**Question5:** Construct a Moore machine that takes binary numbers as input and produces residue modulo '3' as output.

**Solution:**  $\Sigma = \{0,1\}$   
 $\Delta = \{0,1,2\}$



| States            | 0     | 1     | $\Delta$ |
|-------------------|-------|-------|----------|
| $\rightarrow q_0$ | $q_0$ | $q_1$ | 0        |
| $q_1$             | $q_2$ | $q_0$ | 1        |
| $q_2$             | $q_1$ | $q_2$ | 2        |

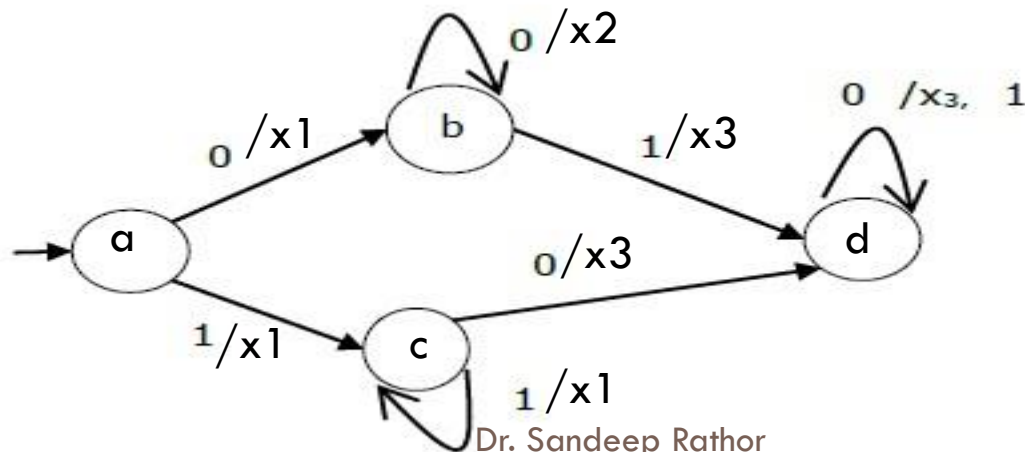
# Mealy Machine

A Mealy machine can be described by a 6 tuple  
i.e.  $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$  where –

- $Q$  is a finite set of states.
- $\Sigma$  is a finite set of symbols called the input alphabet.
- $\Delta$  is a finite set of symbols called the output alphabet.
- $\delta$  is the input transition function where  $\delta: Q \times \Sigma \rightarrow Q$
- $\lambda$  is the output function where  $\lambda : Q \times \Sigma \rightarrow \Delta$
- $q_0$  is the initial state ( $q_0 \in Q$ ).

# Example of Mealy Machine

| Present state | Next state |           |           |           |
|---------------|------------|-----------|-----------|-----------|
|               | input = 0  |           | input = 1 |           |
|               | State      | Output    | State     | Output    |
| → <b>a</b>    | <b>b</b>   | <b>x1</b> | <b>c</b>  | <b>x1</b> |
| <b>b</b>      | <b>b</b>   | <b>x2</b> | <b>d</b>  | <b>x3</b> |
| <b>c</b>      | <b>d</b>   | <b>x3</b> | <b>c</b>  | <b>x1</b> |
| <b>d</b>      | <b>d</b>   | <b>x3</b> | <b>d</b>  | <b>x2</b> |

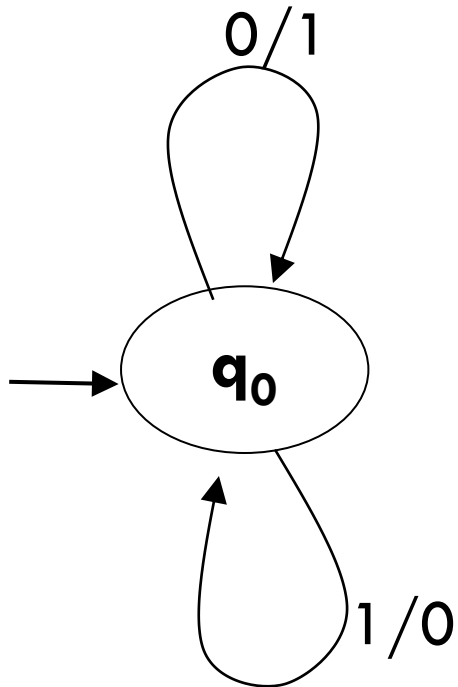


# Construction of Mealy Machine

**Question1:** Construct a mealy machine that takes binary number as input and produces **1's complement** of that number as output. Assume that string is read LSB to MSB.

**Solution:**

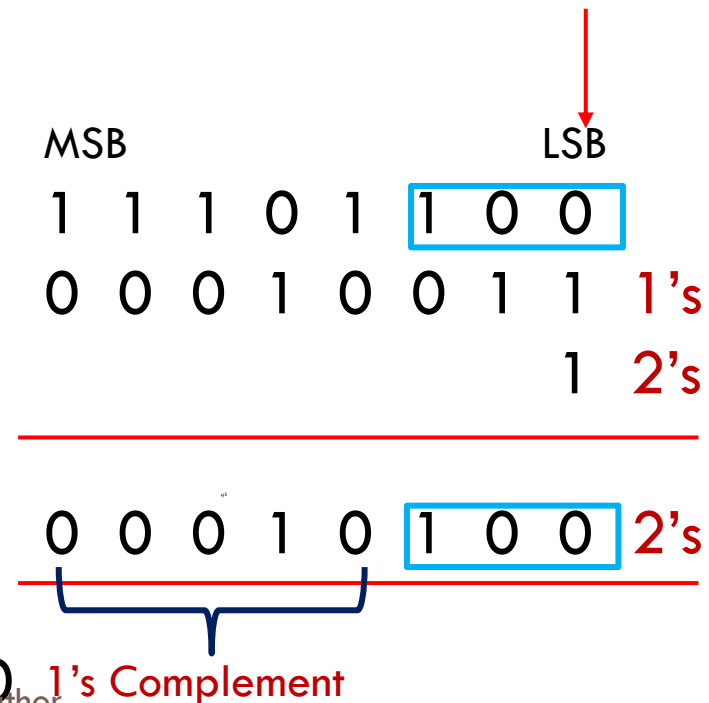
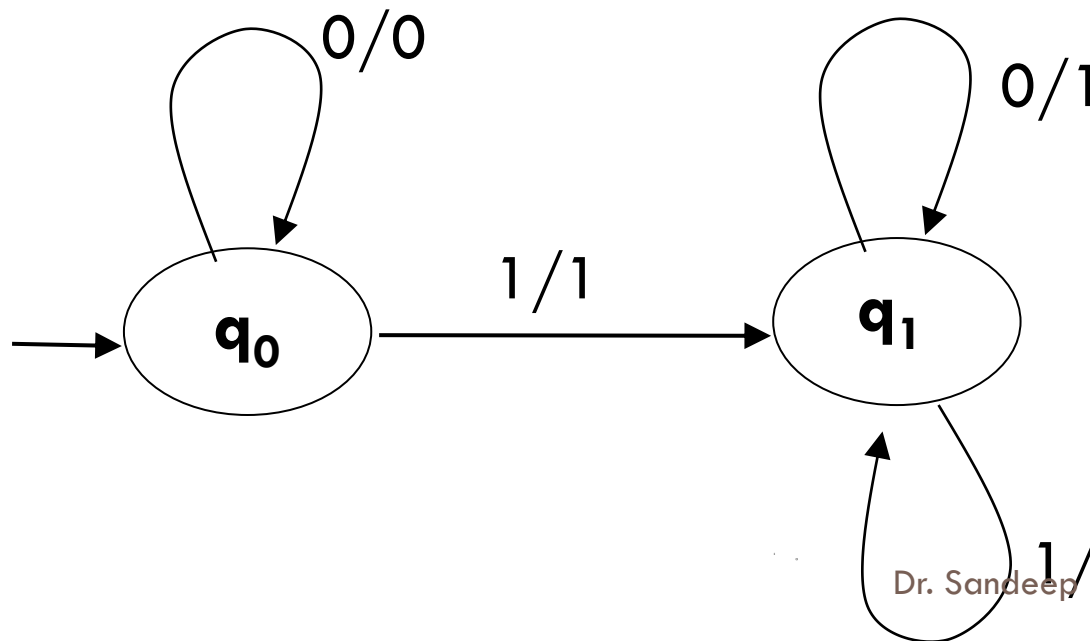
$$\Sigma = \{0,1\}$$
$$\Delta = \{0,1\}$$



# Construction of Mealy Machine

**Question 1:** Construct a mealy machine that takes binary number as input and produces **2's complement** of that number as output. Assume that string is read LSB to MSB and end carry is discarded.

**Solution: Hint**





# Difference b/w Moore & Mealy Machine

| Moore Machine   | Mealy Machine   |
|---|---|
| Output depends only upon the present state.   | Output depends both upon the present state and the present input  |
| Generally, it has more states than Mealy Machine.   | Generally, it has fewer states than Moore Machine.  |
| The value of the output function is a function of the current state and the changes at the clock edges, whenever state changes occur.         | The value of the output function is a function of the transitions and the changes, when the input logic on the present state is done. |
| In Moore machines, more logic is required to decode the outputs resulting in more circuit delays. They generally react one clock cycle later. | Mealy machines react faster to inputs. They generally react in the same clock cycle.  |

# Conversion: Moore to Mealy Machine

## Algorithm

**Input** – Moore Machine

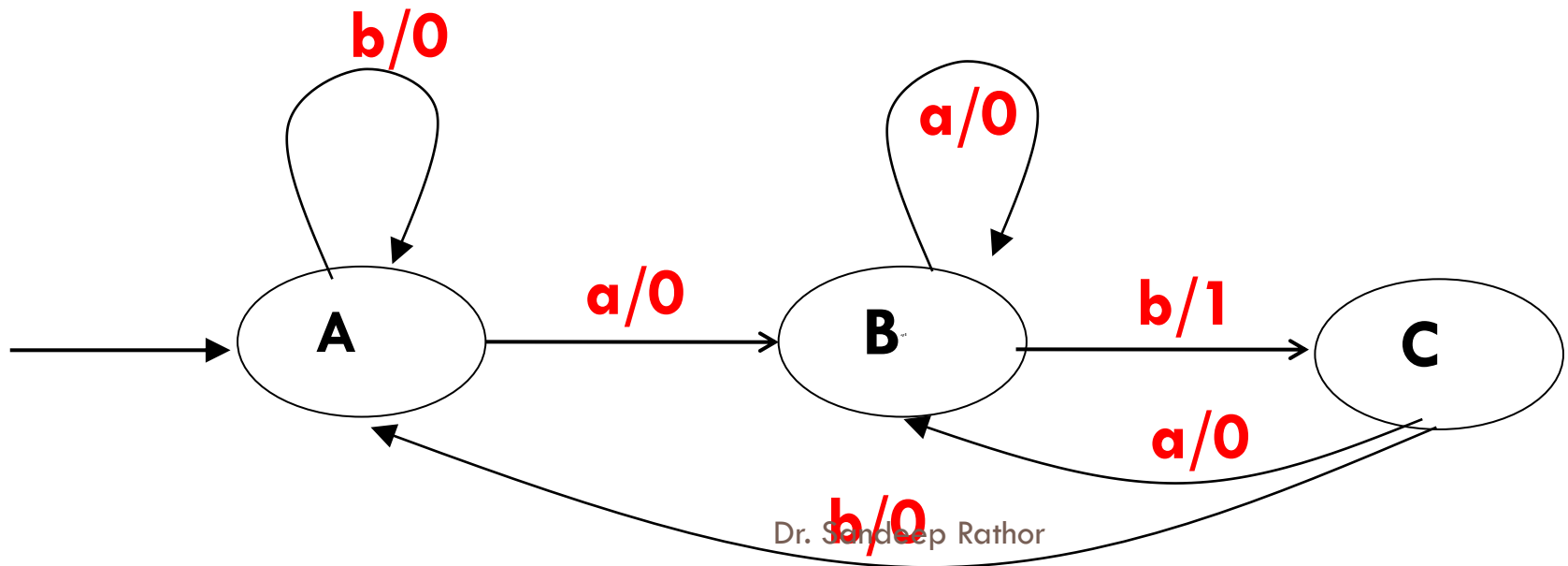
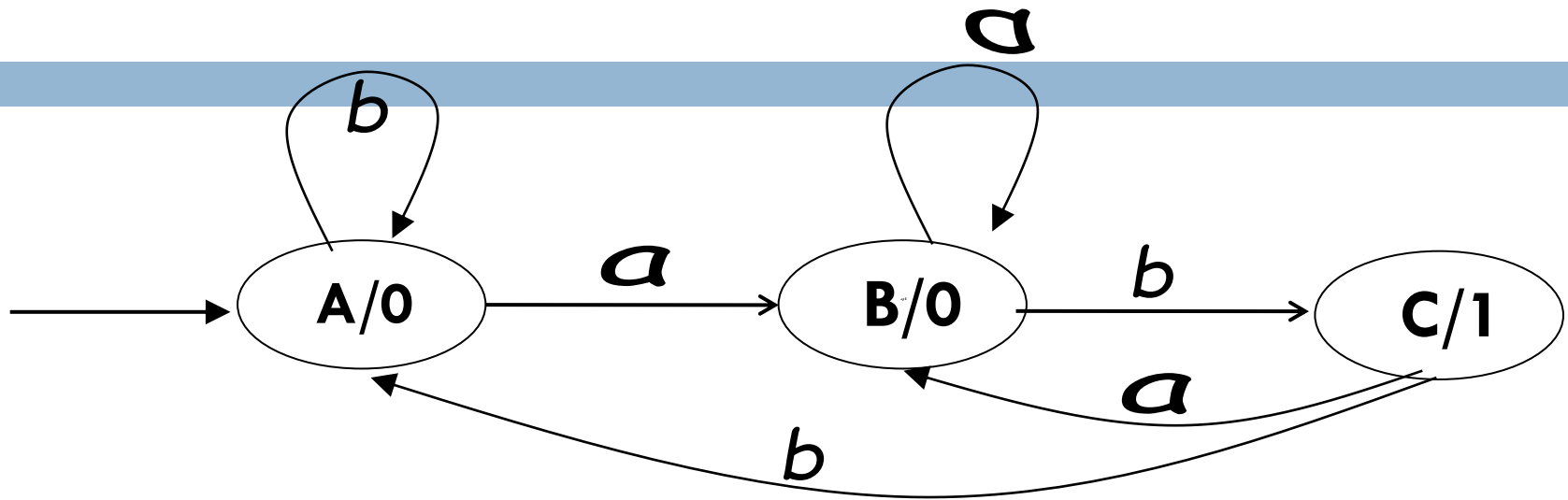
**Output** – Mealy Machine

**Step 1** – Take a blank Mealy Machine transition table format.

**Step 2** – Copy all the Moore Machine transition states into this table format.

**Step 3** – Check the present states and their corresponding outputs in the Moore Machine state table; if for a state  $Q_i$  output is  $m$ , copy it into the output columns of the Mealy Machine state table wherever  $Q_i$  appears in the next state.

# Convert given Moore M/C to Mealy



# Convert given Moore M/C to Mealy

| Present State | Next State |       | Output |
|---------------|------------|-------|--------|
|               | a = 0      | a = 1 |        |
| → a           | d          | b     | 1      |
| b             | a          | d     | 0      |
| c             | c          | c     | 0      |
| d             | b          | a     | 1      |

| Present State | Next State |        |       |        |
|---------------|------------|--------|-------|--------|
|               | a = 0      |        | a = 1 |        |
|               | State      | Output | State | Output |
| => a          | d          | 1      | b     | 0      |
| b             | a          | 1      | d     | 1      |
| c             | c          | 0      | c     | 0      |
| d             | b          | 0      | a     | 1      |

# Conversion: Mealy Machine to Moore

## Algorithm

**Input** – Mealy Machine

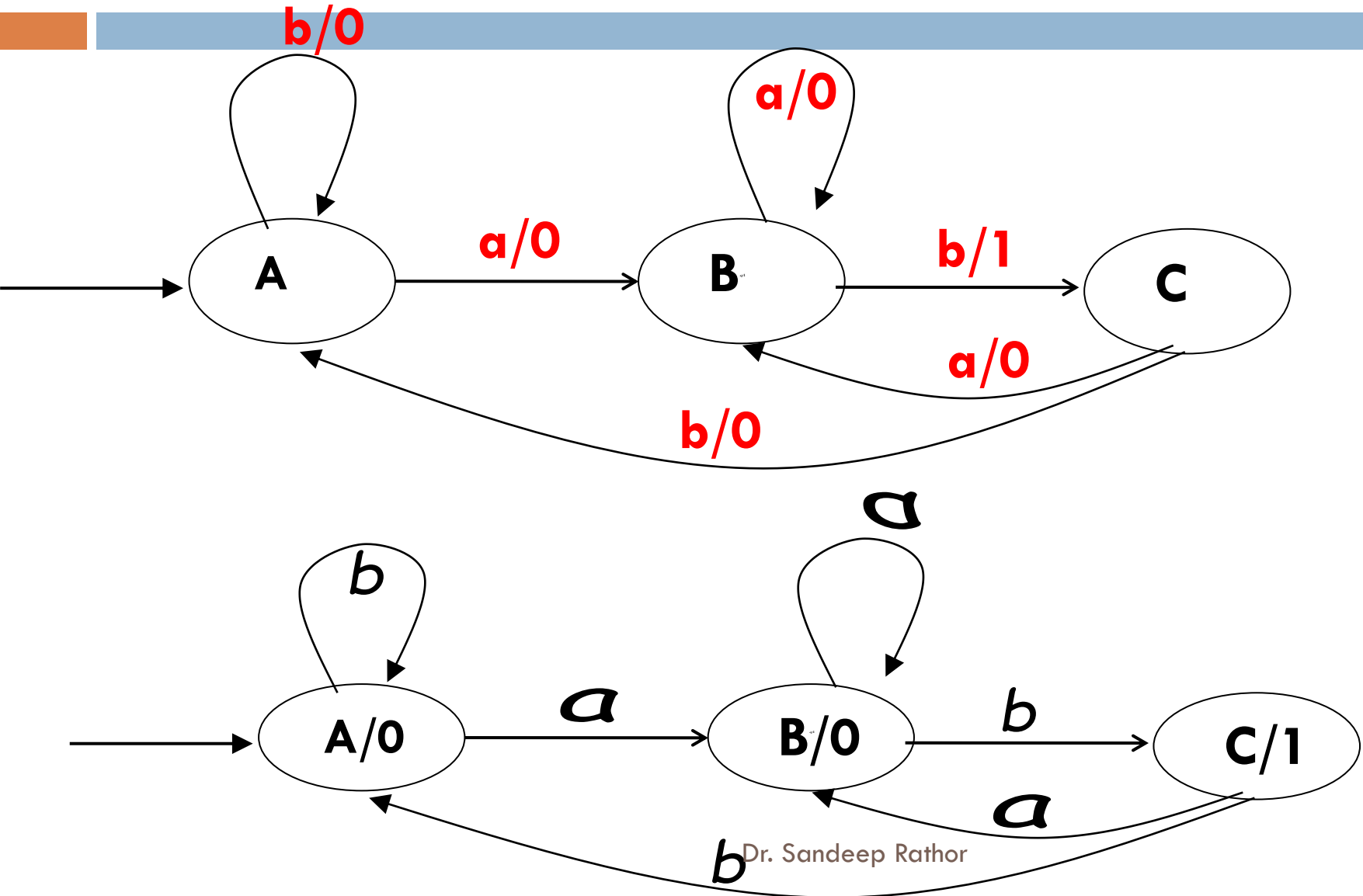
**Output** – Moore Machine

**Step 1** – Calculate the number of different outputs for each state ( $Q_i$ ) that are available in the state table of the Mealy machine.

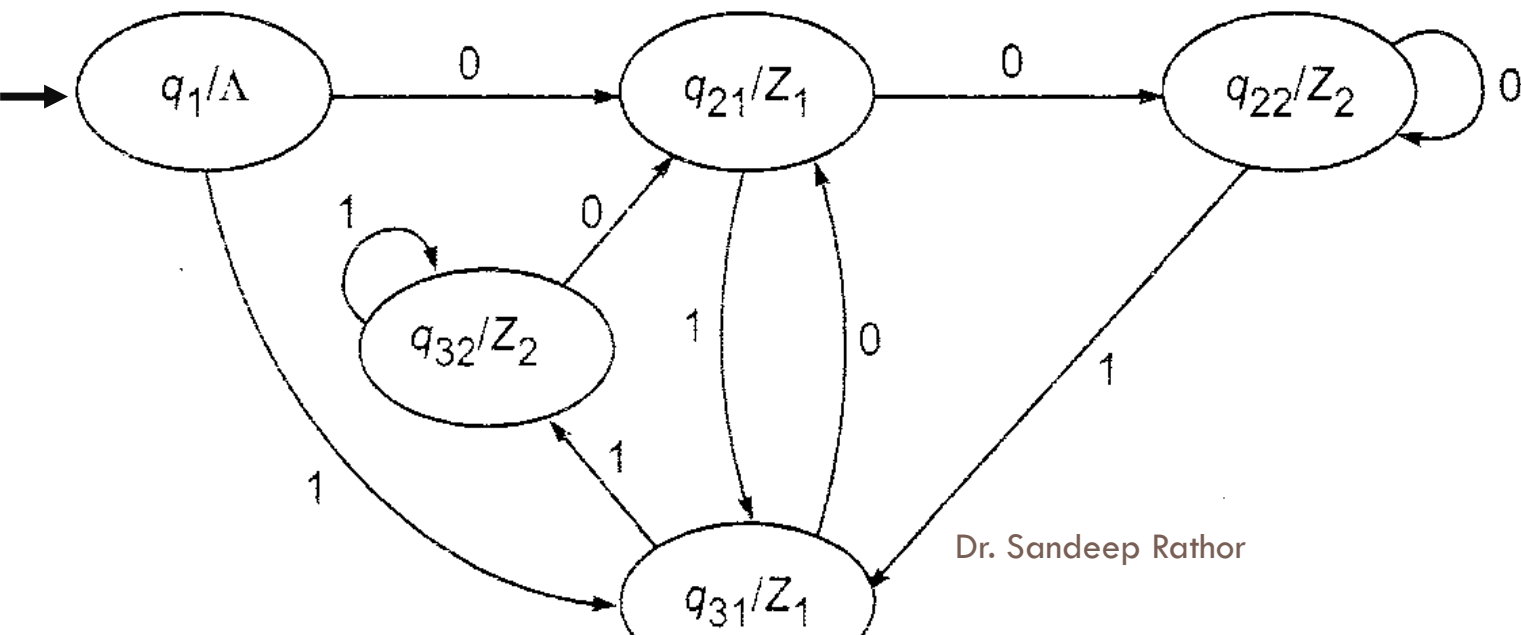
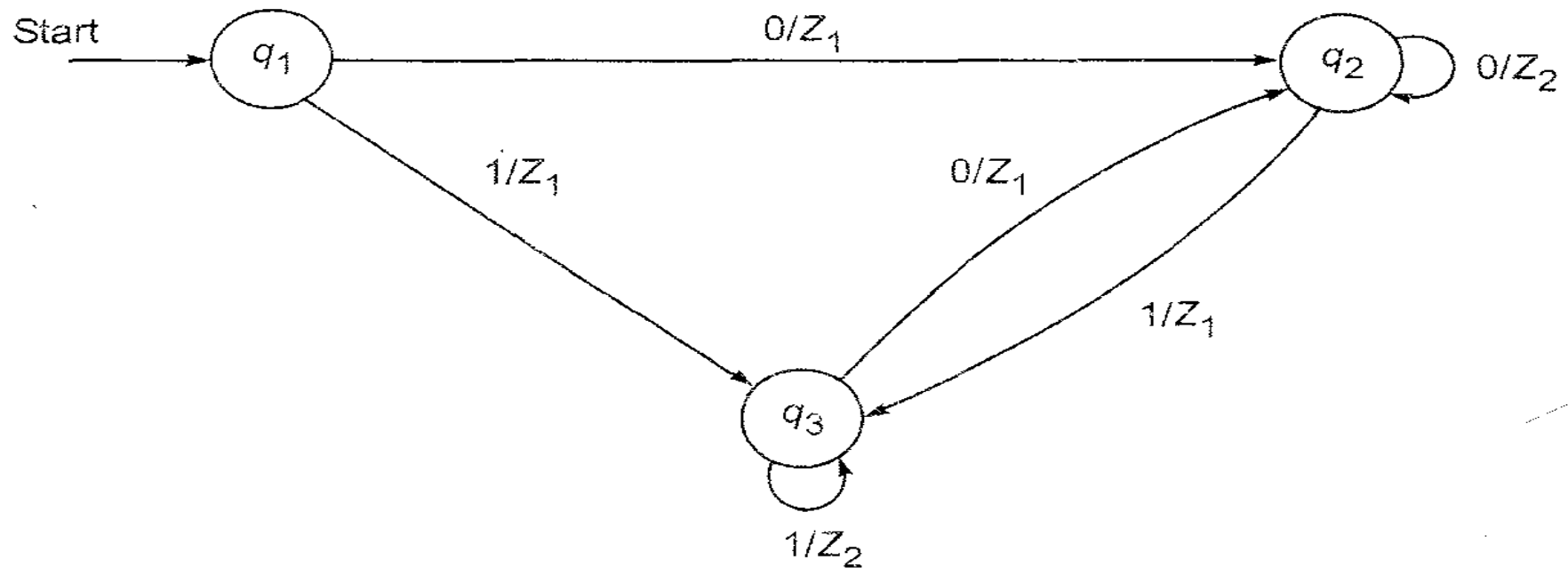
**Step 2** – If all the outputs of  $Q_i$  are same, copy state  $Q_i$ . If it has  $n$  distinct outputs, break  $Q_i$  into  $n$  states as  $Q_{in}$  where  $n = 0, 1, 2, \dots$

**Step 3** – If the output of the initial state is 1, insert a new initial state at the beginning which gives 0 output.

# Convert given Mealy M/C to Moore



# Convert given Mealy M/C to Moore



# Convert given Mealy M/C to Moore

(\*No. of states may be increased)

| Present State | Next State |        |            |        |
|---------------|------------|--------|------------|--------|
|               | a = 0      |        | a = 1      |        |
|               | Next State | Output | Next State | Output |
| → a           | d          | 0      | b          | 1      |
| b             | a          | 1      | d          | 0      |
| c             | c          | 1      | c          | 0      |
| d             | b          | 0      | a          | 1      |

| Present State  | Next State     |                | Output |
|----------------|----------------|----------------|--------|
|                | a = 0          | a = 1          |        |
| → a            | d              | b <sub>1</sub> | 1      |
| b <sub>0</sub> | a              | d              | 0      |
| b <sub>1</sub> | a              | d              | 1      |
| c <sub>0</sub> | c <sub>1</sub> | C <sub>0</sub> | 0      |
| c <sub>1</sub> | c <sub>1</sub> | C <sub>0</sub> | 1      |
| d              | b <sub>0</sub> | a              | 0      |





# REGULAR EXPRESSION

By: Dr. Sandeep Rathor

# Regular Expressions



- RE are used for representing certain sets of string in an algebraic form.
- It describe the language that is accepted by **Finite Automata**.
- The symbols that appear in RE are letters of alphabets  $\Sigma$ , symbol for null string  $\epsilon$ , parenthesis, star operator and plus sign.

A Regular Expression can be recursively defined as:

- $\epsilon$  is a Regular Expression indicates the language containing an empty string. ( $L(\epsilon) = \{\epsilon\}$ )
- $\varphi$  is a Regular Expression denoting an empty language. ( $L(\varphi) = \{ \}$ )

# Regular expressions: Rule

- Union of two RE,  $R_1$  and  $R_2$ , written as  **$R_1 + R_2$** , is also a RE.
- Concatenation of two RE,  $R_1$  and  $R_2$ , written as  **$R_1 R_2$** , is also a RE.
- Iteration of RE,  $R$  written as  **$R^*$** , is also a RE.

# RE Examples



| Regular Expressions            | Regular Set   |
|--------------------------------|---|
| $(0 + 10^*)$                   | $L = \{ 0, 1, 10, 100, 1000, 10000, \dots \}$   |
| $(0^*10^*)$                    | $L = \{1, 01, 10, 010, 0010, \dots\}$   |
| $(0 + \epsilon)(1 + \epsilon)$ | $L = \{\epsilon, 0, 1, 01\}$  |
| $(a+b)^*$                      | Set of strings of a's and b's of any length including the null string.<br>So $L = \{ \epsilon, a, b, aa, ab, bb, ba, aaa, \dots \}$   |
| $(a+b)^*abb$                   | Set of strings of a's and b's ending with the string abb. So $L = \{abb, aabb, babb, aaabb, ababb, \dots\}$   |
| $(11)^*$                       | Set consisting of even number of 1's including empty string, So $L = \{\epsilon, 11, 1111, 111111, \dots\}$   |
| $(aa)^*(bb)^*b$                | Set of strings consisting of even number of a's followed by odd number of b's, so $L = \{b, aab, aabbb, aabbbbb, aaaab, aaaabbb, \dots\}$   |
| $(aa + ab + ba + bb)^*$        | String of a's and b's of even length can be obtained by concatenating any combination of the strings aa, ab, ba and bb including null, so $L = \{aa, ab, ba, bb, aaab, aaba, \dots\}$ |

# Represent these sets by RE



$\{\epsilon, 0, 00, 000, 0000, \dots\}$

Ans:  $0^*$

$\{\epsilon, ab\}$

Ans:  $\epsilon + ab$

$\{01, 10\}$

Ans:  $01 + 10$

$\{\epsilon, 10, 01\}$

Ans:  $\epsilon + 10 + 01$

Set of all strings ending in b.

Ans:  $(a+b)^*b$

Set of all strings starting with a and ending with ba

Ans:  $a(a+b)^*ba$

(a) The set of all strings over  $\{0, 1\}$  with three consecutive 0's.

$$(0+1)^*000(0+1)^*$$

(b) The set of all strings over  $\{0, 1\}$  beginning with 00.

$$00(0+1)^*$$

(c) The set of all strings over  $\{0, 1\}$  ending with 00 and beginning with 1.

$$1(0+1)^*00$$

(d) all the string containing exactly two 0's.

$$1^*01^*01^*$$

(e)

- All strings containing an even number of 0's:

$$1^* + (1^*01^*0)^*1^*$$

- All strings having at least two occurrences of the substring 00:

$$(1 + 0)^* 00(1 + 0)^* 00(1 + 0)^* + (1 + 0)^* 000(1 + 0)^*$$

Find a regular expression corresponding to the language of strings of even lengths over the alphabet of  $\{a, b\}$ .

$$(aa + ab + ba + bb)^*$$



- Find a regular expression corresponding to the language of all strings over the alphabet  $\{ a, b \}$  that do not end with  $ab$ .

**$(a + b)^*(a + bb)$**

- Find a regular expression corresponding to the language of all strings over the alphabet  $\{ a, b \}$  that contain exactly two  $a$ 's.

□  **$b^*ab^*ab^*$**

# Identities Related to Regular Expressions

Given R, P, L, Q as regular expressions, the following identities hold

$$\emptyset^* = \varepsilon$$

$$\varepsilon^* = \varepsilon$$

$$RR^* = R^*R$$

$$R^*R^* = R^*$$

$$(R^*)^* = R^*$$

$$(PQ)^*P = P(QP)^*$$

$$(a+b)^* = (a^*b^*)^* = (a^*+b^*)^* = (a+b^*)^* = a^*(ba^*)^*$$

$$R + \emptyset = \emptyset + R = R \text{ (The identity for union)}$$

$$R \varepsilon = \varepsilon R = R \text{ (The identity for concatenation)}$$

$$\emptyset L = L \emptyset = \emptyset \text{ (The annihilator for concatenation)}$$

$$R + R = R \text{ (Idempotent law)}$$

$$L(M + N) = LM + LN \text{ (Left distributive law)}$$

$$(M + N)L = ML + NL \text{ (Right distributive law)}$$

$$\varepsilon + RR^* = \varepsilon + R^*R = R^*$$

# Arden's Theorem

# Arden's Theorem



**Statement:** Let  $P$  and  $Q$  are two regular expressions over  $\Sigma$  and if  $P$  does not contain  $\epsilon$ , then the following equation in  $R$ ,  $R = Q + RP$  has a unique solution given by

$$R = QP^*$$

**Application:** The Arden's Theorem is useful to solve a regular expression.

# Proof

**Part I: Prove that  $R = QP^*$  is the solution of this equation**

$$R = Q + RP$$

Replace  $R$  by  $QP^*$  on both sides

$$\text{LHS} = QP^*$$

$$\text{RHS} = Q + QP^*P$$

$$= Q (\varepsilon + P^*P)$$

$$= QP^* \quad // \text{ (As we know that } \varepsilon + A^*A = A^*)$$

$$=\text{LHS}$$

Thus,  $R = QP^*$  is the solution of the equation  $R = Q + RP$ .

## Part II : Prove that this is the only solution of this equation.



$$R = Q + RP$$

Replace R by  $Q + RP$  on RHS

$$\begin{aligned} R &= Q + (Q + RP) P \\ &= Q + QP + RP^2 \end{aligned}$$

Keep replacing R by  $Q + RP$

$$\begin{aligned} R &= Q + QP + (Q + RP) P^2 \\ &= Q + QP + QP^2 + RP^3 \\ &\dots \end{aligned}$$

$$= Q + QP + QP^2 + QP^3 + \dots + QP^i + RP^{i+1}$$

$$\begin{aligned} R &= Q + QP + QP^2 + QP^3 + \dots + QP^i + RP^{i+1} \\ &= Q (\varepsilon + P + P^2 + \dots + P^i) + RP^{i+1} \quad \text{for } i \geq 0 \end{aligned}$$

We claim that any soln of  $R = Q + RP$  must be equivalent to  $QP^*$

Let  $w \in R$  and  $|w| = i$ .

then  $w$  belongs to set  $Q (\varepsilon + P + P^2 + \dots + P^i) + RP^{i+1}$ ,

as  $P$  does not contain null.  $RP^{i+1}$  has no string of length less  $i+1$  so,

$w$  is not in set  $RP^{i+1}$

It means  $w$  belongs to the set  $Q (\varepsilon + P + P^2 + \dots + P^i)$

and hence it is equivalent to  $QP^*$

# Arden's Theorem

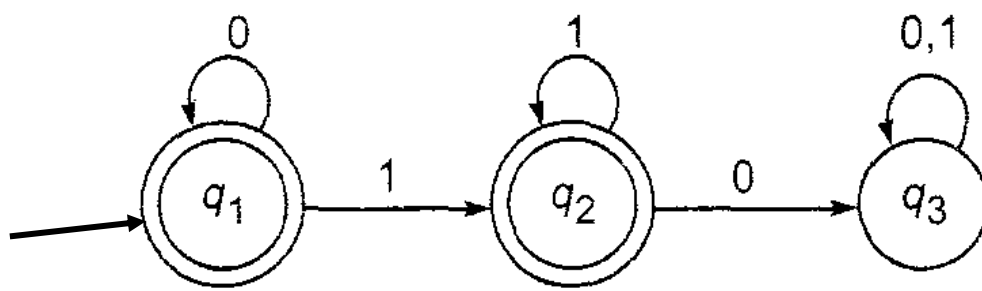
## DFA to RE



# Assumptions for Applying Arden's Theorem



- The transition diagram must not have NULL transitions
- It must have only one initial state



Find Regular Expression of the given DFA?

$$q_1 = q_1 0 + \Lambda$$

$$q_2 = q_1 1 + q_2 1$$

$$q_3 = q_2 0 + q_3 (0 + 1)$$

$$q_1 = \Lambda 0^* = 0^*$$

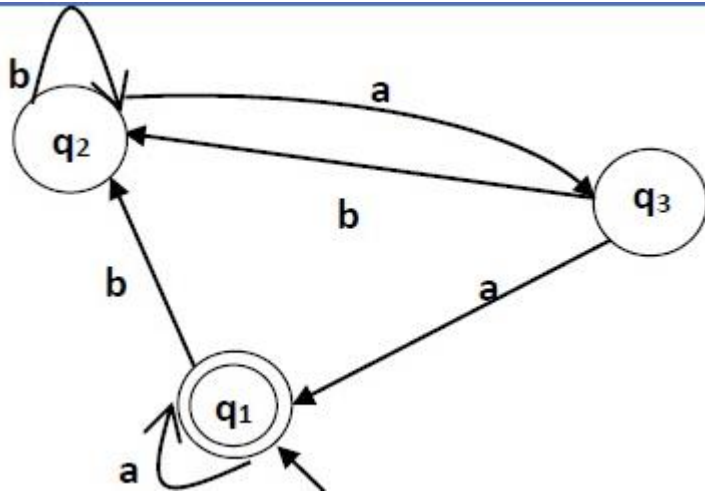
Using Arden's Theorem

$$q_2 = q_1 1 + q_2 1 = 0^* 1 + q_2 1$$

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

$$q_1 + q_2 = 0^* + 0^*(11^*) = 0^*(\Lambda + 11^*) = 0^*(1^*)$$

Find the RE



### Step 1: Construct the equations

$$q_1 = q_1a + q_3a + \varepsilon$$

$$q_2 = q_1b + q_2b + q_3b$$

$$q_3 = q_2a$$

### Step 2: Solve the equations

$$(a + b(b + ab)^*aa)^*$$

$$q_1 = q_1a + q_3a + \epsilon$$

$$q_2 = q_1b + q_2b + q_3b$$

$$q_3 = q_2a$$

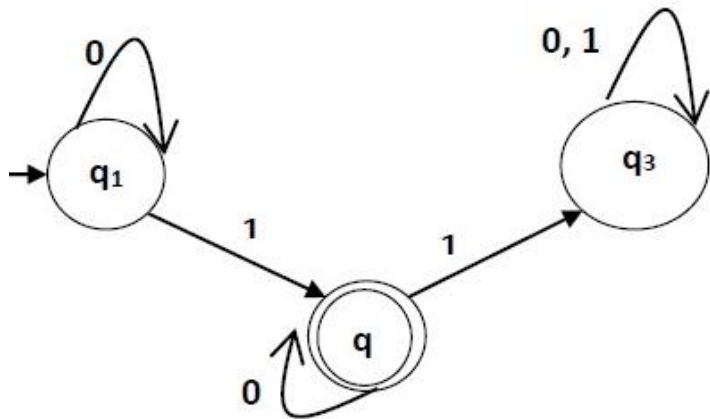
Now, we will solve these three equations –

$$\begin{aligned} q_2 &= q_1b + q_2b + q_3b \\ &= q_1b + q_2b + (q_2a)b \text{ (Substituting value of } q_3) \\ &= q_1b + q_2(b + ab) \\ &= q_1b (b + ab)^* \text{ (Applying Arden's Theorem)} \end{aligned}$$

$$\begin{aligned} q_1 &= q_1a + q_3a + \epsilon \\ &= q_1a + q_2aa + \epsilon \text{ (Substituting value of } q_3) \\ &= q_1a + q_1b(b + ab^*)aa + \epsilon \text{ (Substituting value of } q_2) \\ &= q_1(a + b(b + ab)^*aa) + \epsilon \\ &= \epsilon (a + b(b + ab)^*aa)^* \\ &= (a + b(b + ab)^*aa)^* \end{aligned}$$

Hence, the regular expression is  **$(a + b(b + ab)^*aa)^*$** .

## Find the RE



### Step 1: Construct the equations

$$q_1 = q_1 0 + \epsilon$$

$$q_2 = q_1 1 + q_2 0$$

$$q_3 = q_2 1 + q_3 (0 + 1)$$

### Step 2: Solve the equations

$$0^*10^*$$

## Solution –

Here the initial state is  $q_1$  and the final state is  $q_2$

Now we write down the equations –

$$q_1 = q_1 0 + \epsilon$$

$$q_2 = q_1 1 + q_2 0$$

$$q_3 = q_2 1 + q_3 0 + q_3 1$$

Now, we will solve these three equations –

$$q_1 = \epsilon 0^* \text{ [As, } \epsilon R = R]$$

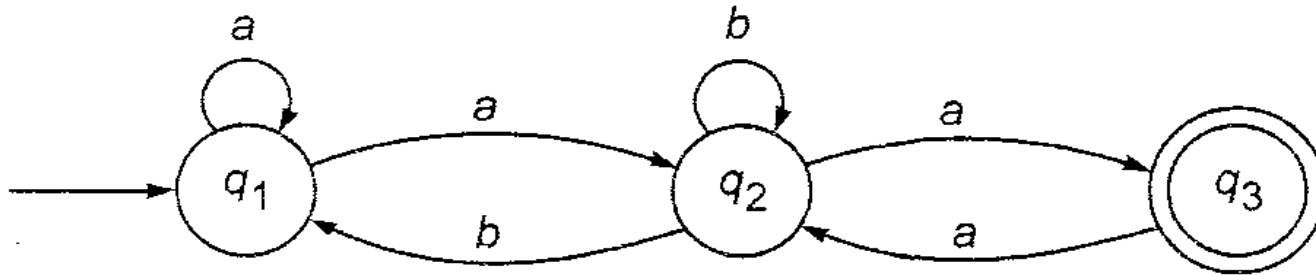
$$\text{So, } q_1 = 0^*$$

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

$$\text{So, } q_2 = 0^* 1 (0)^* \text{ [By Arden's theorem]}$$

Hence, the regular expression is  $0^* 1 0^*$ .

# For Practice



$$q1 = q_1 a + q_2 b + \varepsilon$$

$$q2 = q_1 a + q_2 b + q_3 a$$

$$q3 = q_2 a$$

$$q2 = q_1 a + q_2 b + q_2 a a$$

$$= q_1 a + q_2 (b + a a)$$

$$= q_1 a (b + a a)^*$$

$$q_1 = q_1 a + q_1 a (b + a a)^* b + \varepsilon$$

$$= q_1 (a + a (b + a a)^* b) + \varepsilon$$

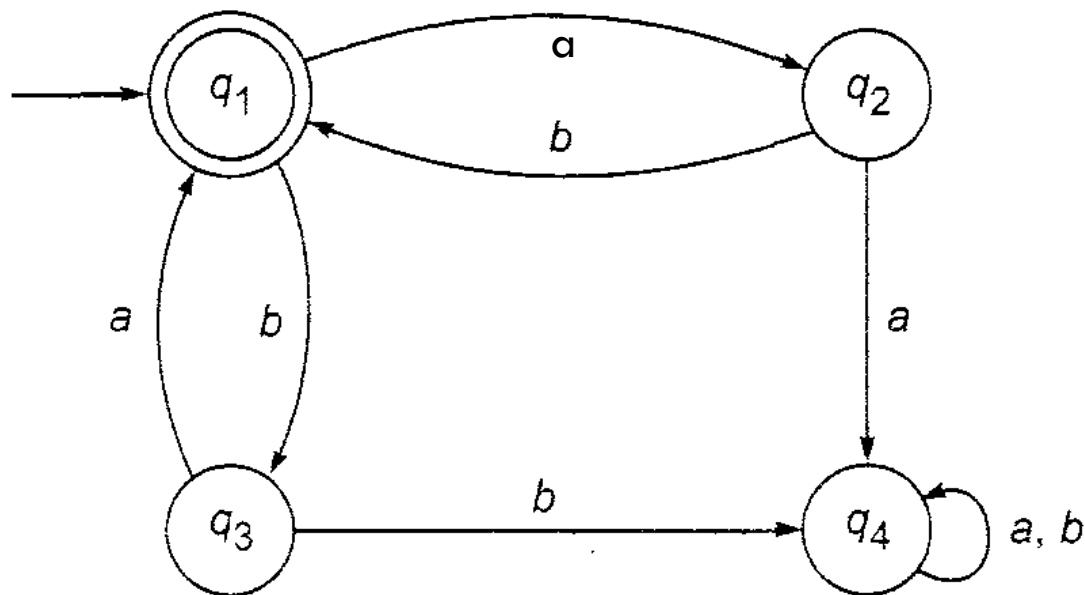
$$q_1 = \varepsilon (a + a (b + a a)^* b)^*$$

$$q2 = (a + a (b + a a)^* b)^* a (b + a a)^*$$

$$q3 = (a + a (b + a a)^* b)^* a (b + a a)^* a$$

Q3 is the final state. So RE

$$(a + a (b + a a)^* b)^* a (b + a a)^* a$$



$$q_1 = q_2b + q_3a + \Lambda$$

$$q_2 = q_1a$$

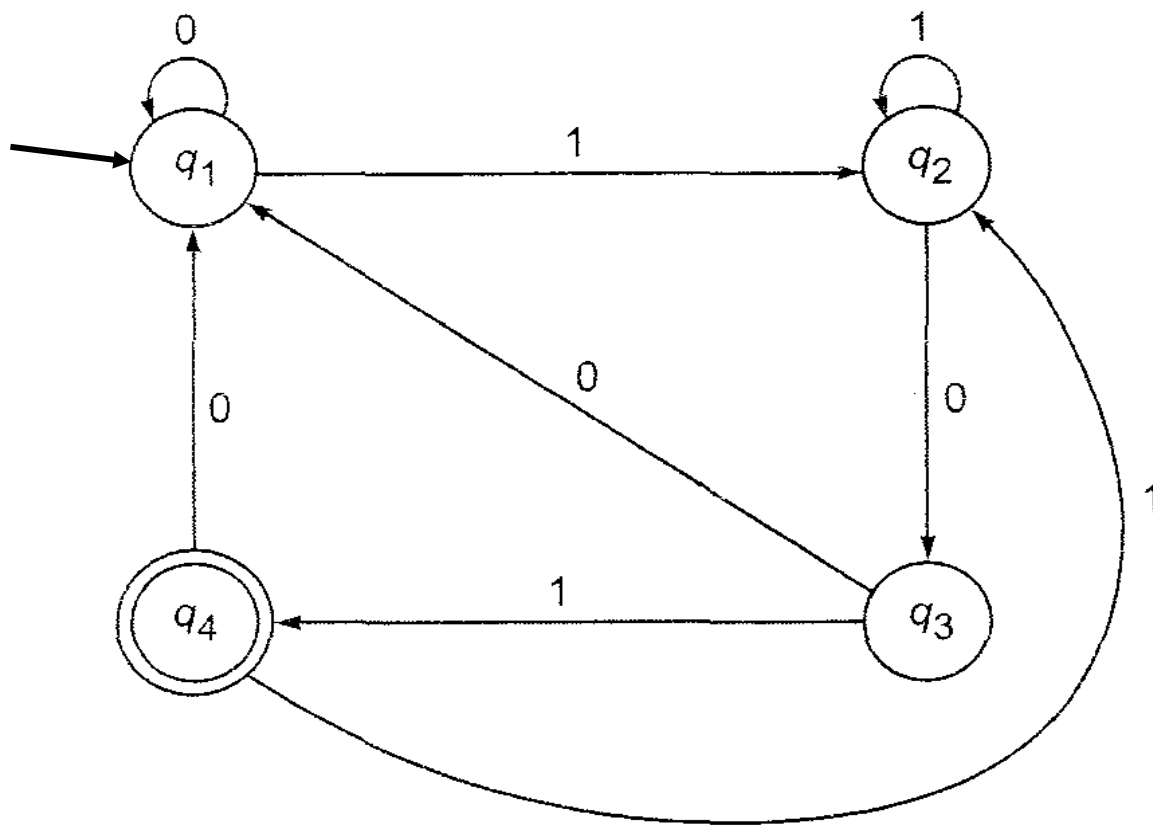
$$q_3 = q_1b$$

$$q_4 = q_2a + q_3b + q_4a + q_4b$$

$$q_1 = q_1ab + q_1ba + \Lambda = q_1(ab + ba) + \Lambda$$

$$q_1 = \Lambda(ab + ba)^* = (ab + ba)^*$$





$$q_1 = q_1 0 + q_3 0 + q_4 0 + \Lambda$$

$$q_2 = q_1 1 + q_2 1 + q_4 1$$

$$q_3 = q_2 0$$

$$q_4 = q_3 1$$

$$q_4 = q_3 1 = (q_2 0) 1 = q_2 0 1$$

$$q_2 = q_11 + q_21 + q_2011 = q_11 + q_2(1 + 011)$$

$$q_2 = (q_11)(1 + 011)^* = q_1(1(1 + 011)^*)$$

$$q_1 = q_10 + q_200 + q_2010 + \Lambda$$

$$= q_10 + q_2(00 + 010) + \Lambda$$

$$= q_10 + q_11(1 + 011)^* (00 + 010) + \Lambda$$

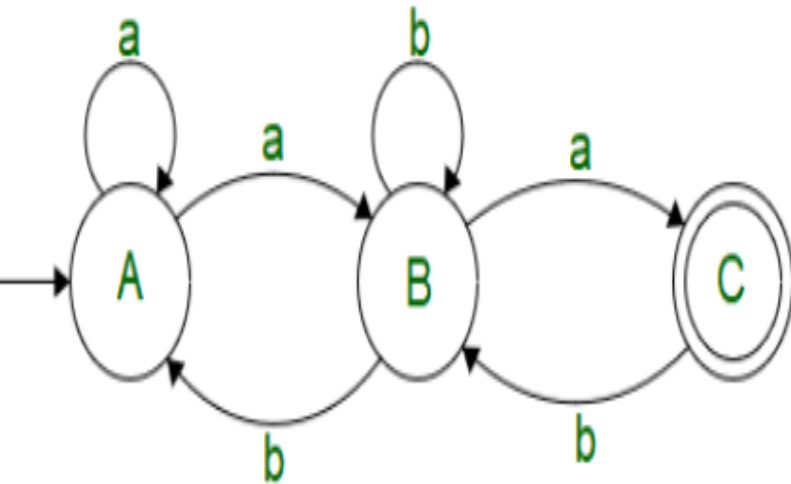
$$q_1 = \Lambda(0 + 1(1 + 011)^* (00 + 010))^*$$

Using Arden's theorem

$$q_4 = q_201 = q_11(1 + 011)^* 01$$

$$= (0 + 1(1 + 011)^*(00 + 010))^*(1(1 + 011)^* 01)$$

Find the RE



### Step 1: Construct the equations

$$A = Aa + Bb + \epsilon$$

$$B = Aa + Bb + Cb$$

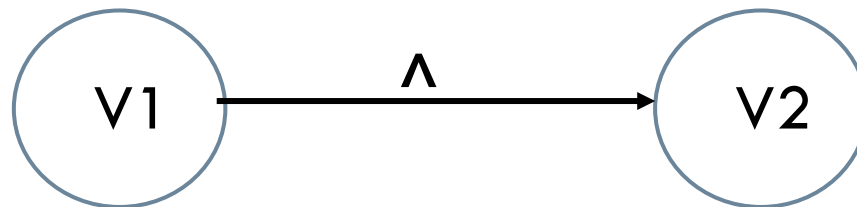
$$C = Ba$$

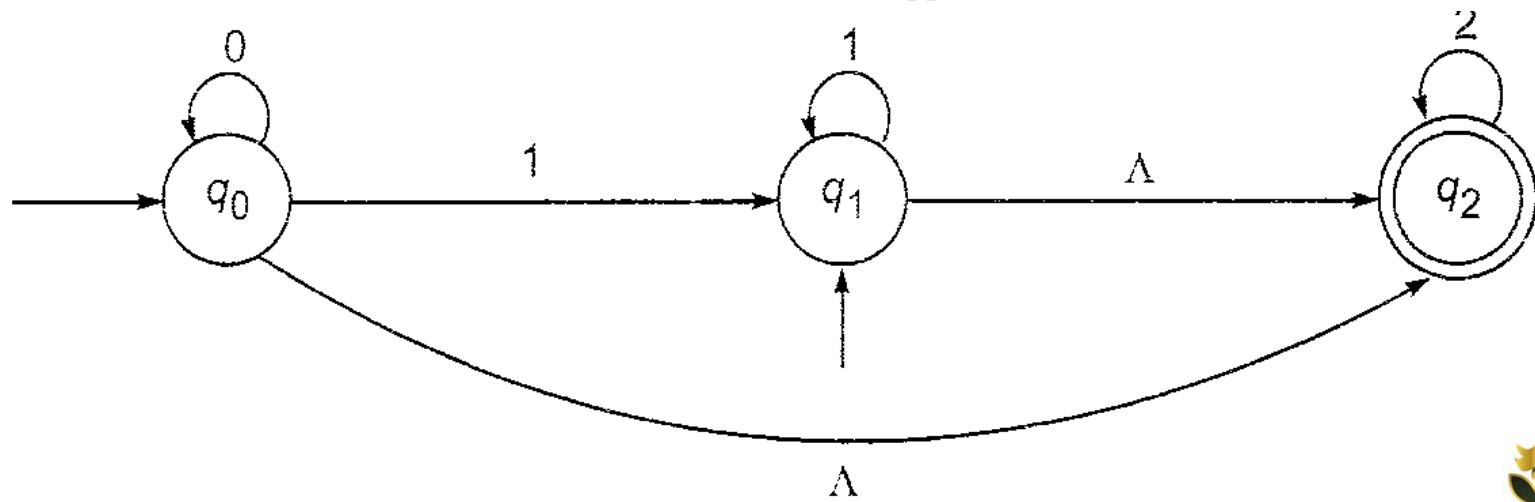
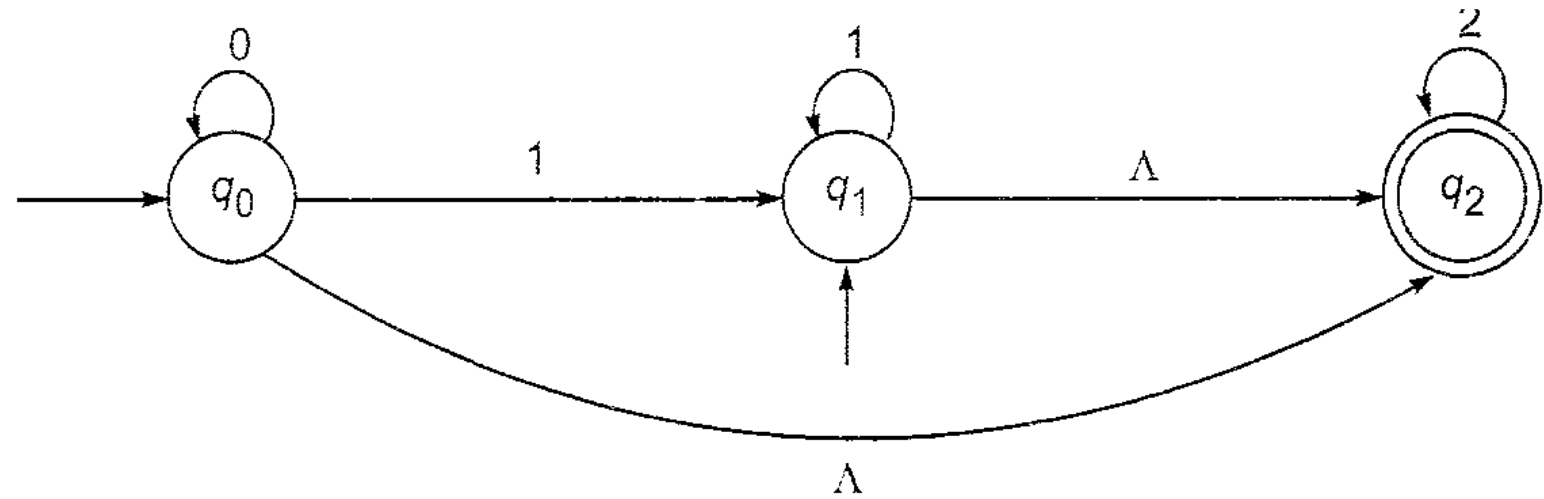
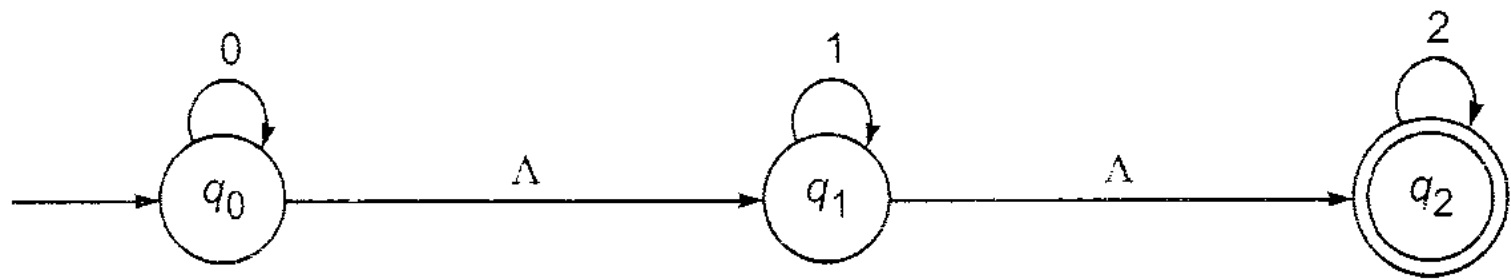
### Step 2: Solve the equations

$$(a + a(b + ab)^*b)^* a (b + ab)^* a$$

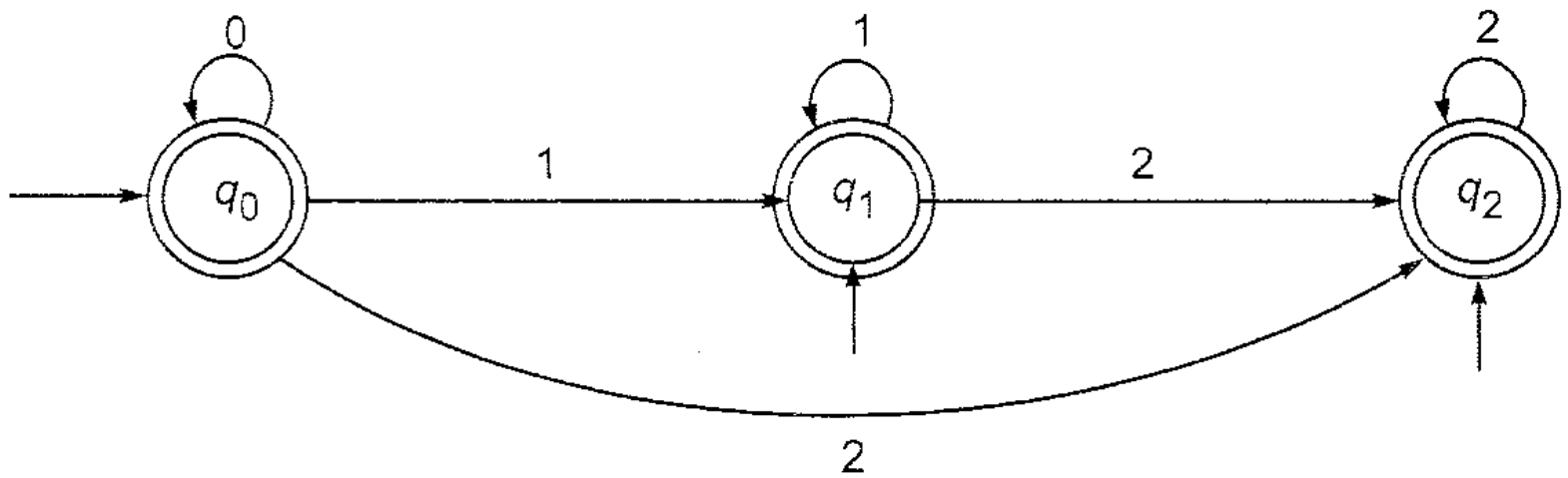
# Elimination of Null Moves

- **Step 1** Find all the edges starting from  $V2$ .
- **Step 2** Duplicate all these edges starting from  $V1$  without changing the edge labels.
- **Step 3** If  $V1$  is an initial state, make  $V2$  also as initial state.
- **Step 4** If  $V2$  is a final state. make  $V1$  also as the final state.





Dr. Sandeep Rathor

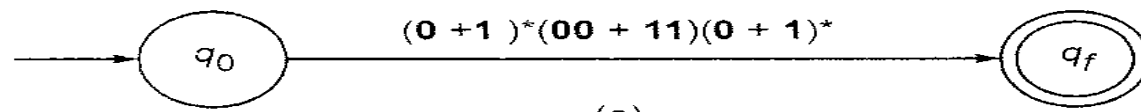


Dr. Sandeep Rathor

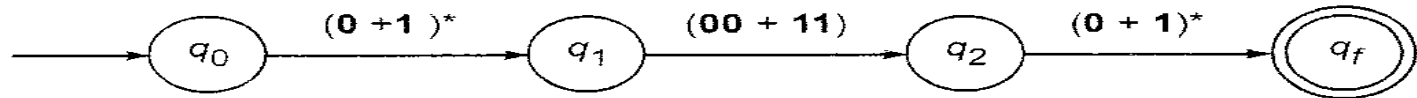
# RE to DFA



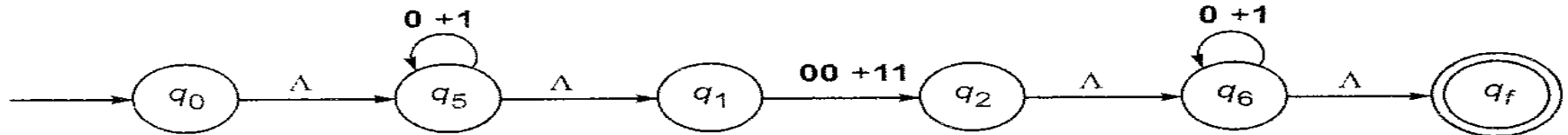
$(0 + 1)^*(00 + 11)(0 + 1)^*$



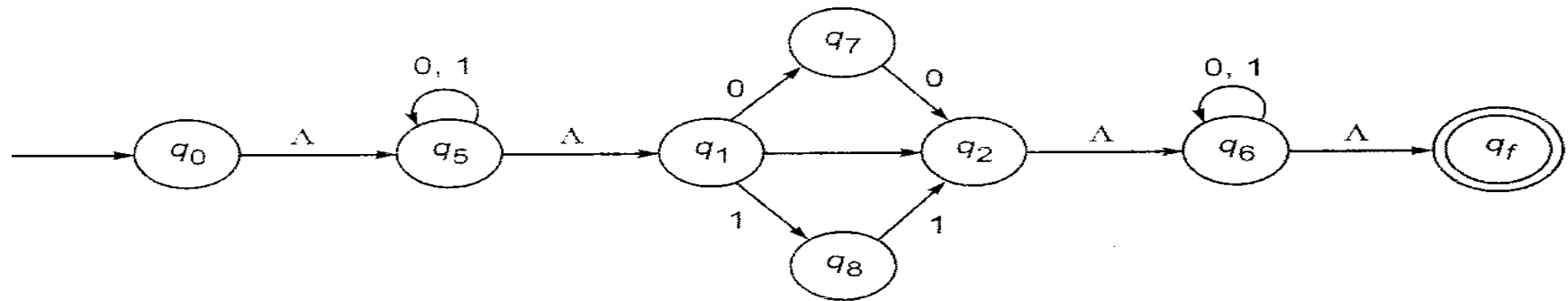
(a)



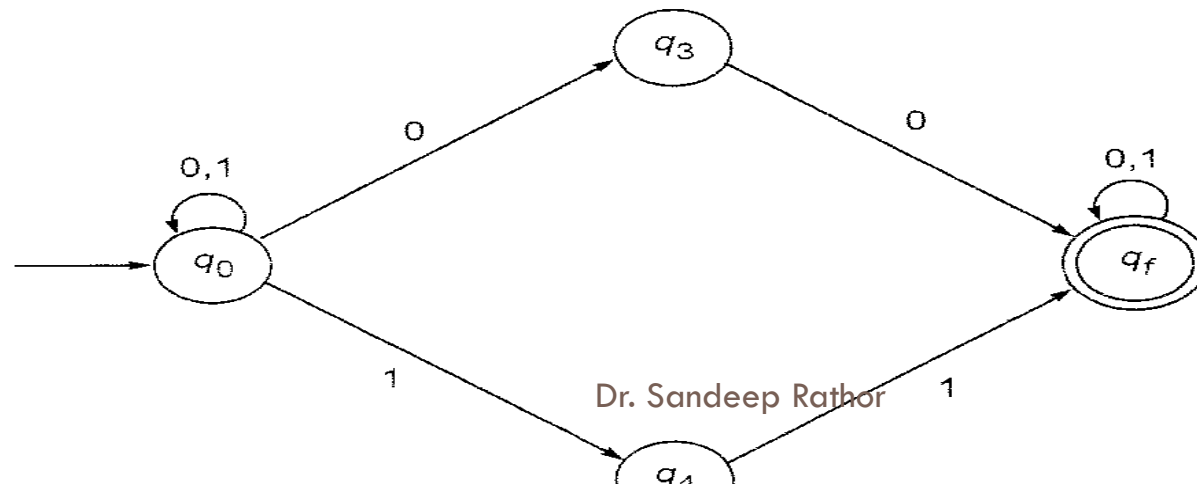
(b)



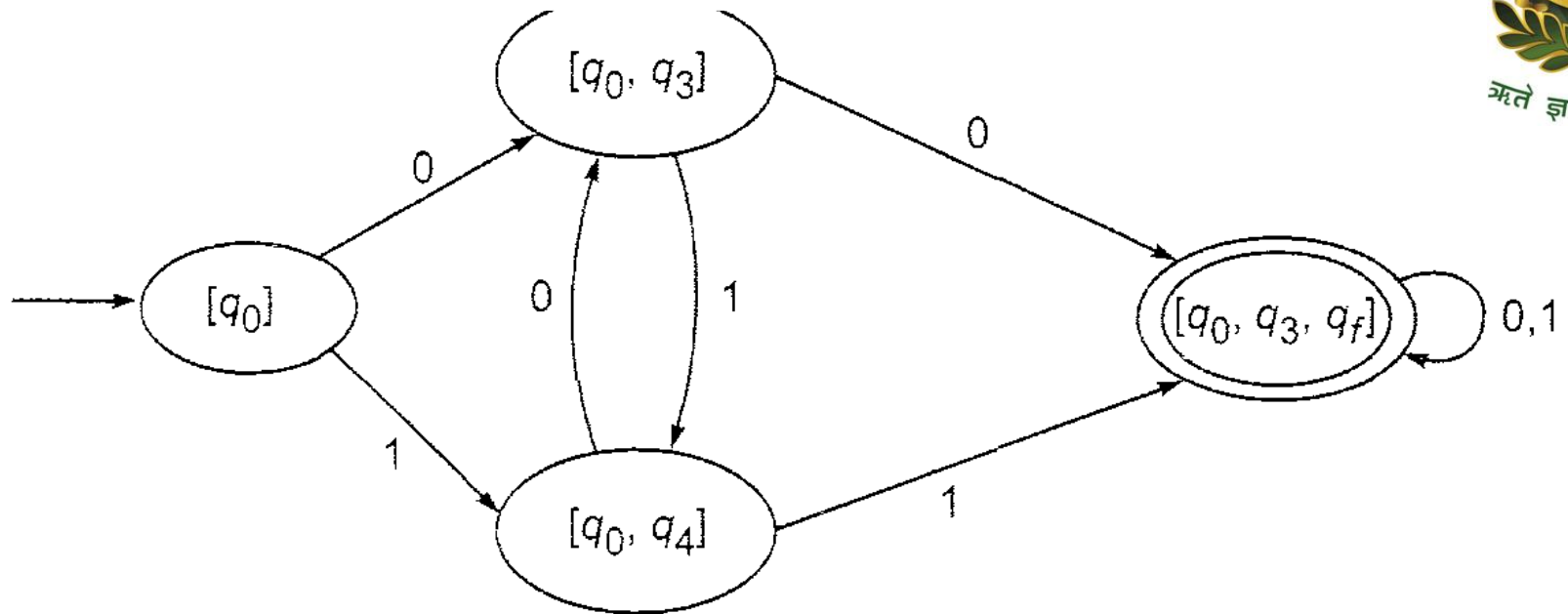
(c)



(d)



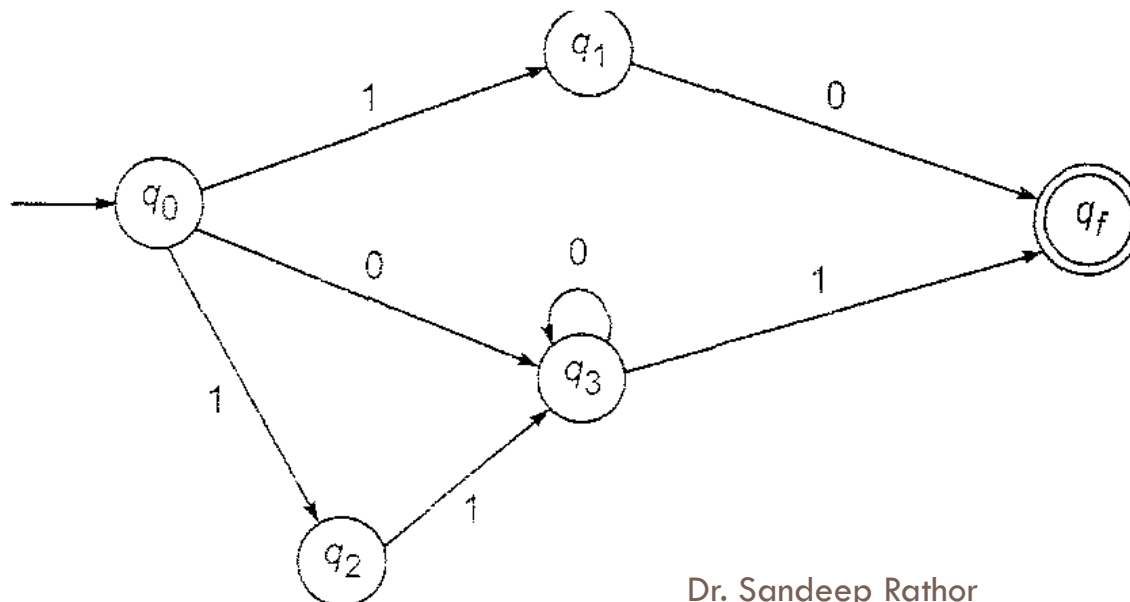
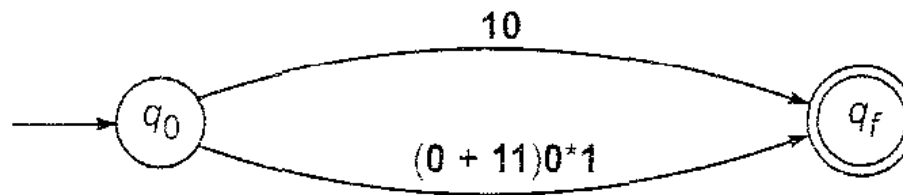


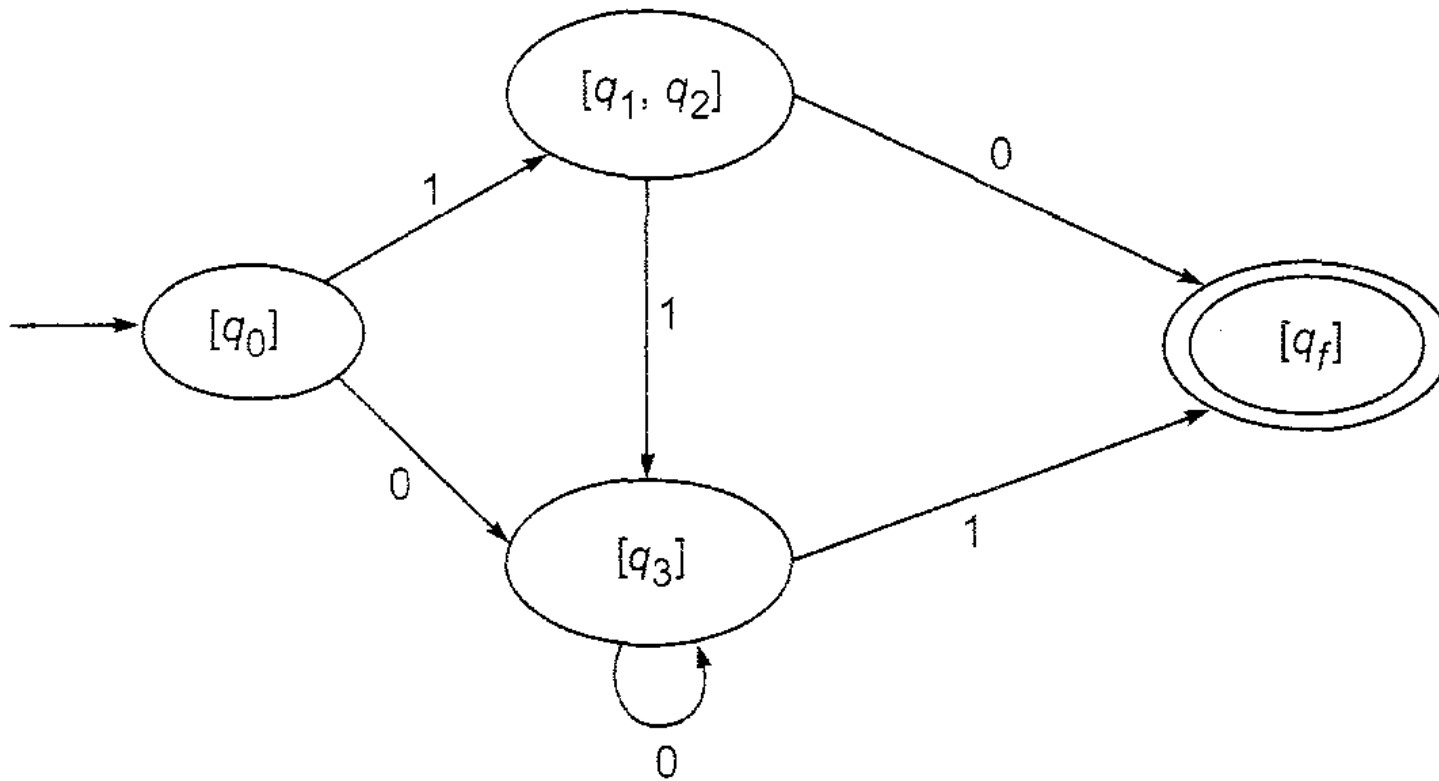


DFA corresponding to given RE

Find DFA from given RE

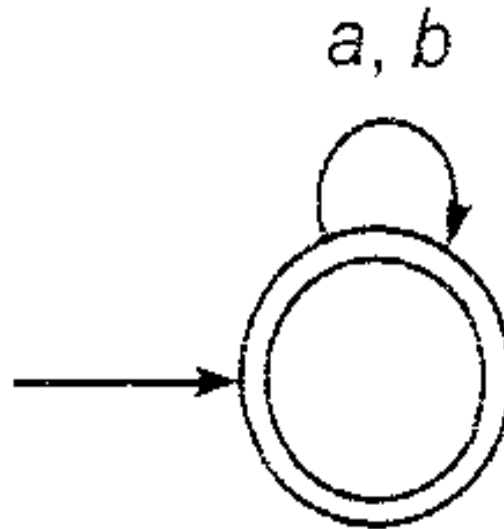
$10 + (0 + 11)0^*1$





Final DFA as per the given RE

# Find DFA of $a^*(ba^*)^*$



# Pumping lemma for Regular Sets



## Statement:

Let  $M = (Q, \Sigma, \delta, q_0, F)$  be a finite automaton with  $n$  states. Let  $L$  be the regular set accepted by  $M$ . Let  $w \in L$  and  $|w| \geq m$ . If  $m \geq n$ , then there exists  $x, y, z$  such that  $w = xyz$ ,  $y \neq \epsilon$ , and  $xy^iz \in L$  for each  $i \geq 0$ .

**\*Pumping** Repeating a section of the string an arbitrary number of times ( $\geq 0$ ), with the resulting string remaining in the language.

# OR



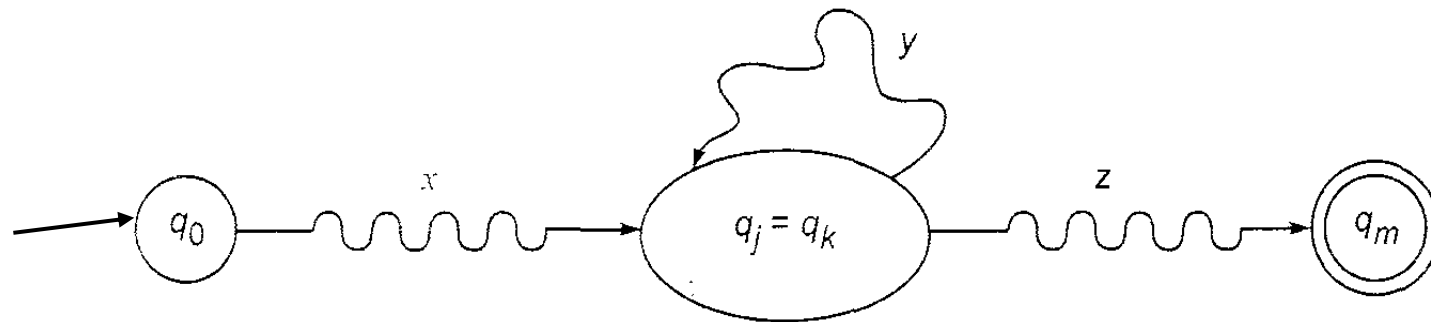
If  $L$  is a Regular Language, then there is a number  $p$  (the pumping length) where if  $w$  is any string in  $L$  of length at least  $p$ , then  $w$  may be divided into 3 pieces,  $w = xyz$ , satisfying the following conditions:

- a. For each  $i \geq 0$ ,  $xy^iz \in L$ ,
- b.  $|y| > 0$ , and
- c.  $|xy| \leq p$ .

# Proof:

- Let  $w = a_1, a_2, \dots, a_m$ ,  $m \geq n$
- $\delta(q_0, a_1, a_2, a_3, \dots, a_i) = q_i$  for  $i = 1, 2, 3, \dots, m$
- $Q = \{q_0, q_1, q_2, \dots, q_m\}$
- $Q$  is the sequence of states in the path with path value  $w = a_1, a_2, a_3, \dots, a_m$ . As there are only  $n$  distinct states, at least two states in  $Q$  must coincide. Among various pair of repeated states, we take the first pair. Let take them as  $q_j$  and  $q_k$  ( $q_j = q_k$ ).  $j$  and  $k$  satisfy the condition  $0 \leq j < k \leq n$ .

- String  $w$  can be decomposed into 3-substrings,
- $x = a_1, a_2, \dots, a_j$
- $Y = a_{j+1} \dots a_k$
- $Z = a_{k+1} \dots a_m$
- The path with the path value  $w$  in the transition diagram of  $M$  is shown as:



Automaton  $M$  starts from the initial state  $q_0$ . On applying the string  $x$ , it reaches  $q_j (= q_k)$ . On applying the string  $y$ , it comes back to  $q_j (= q_k)$ . So, after application of  $y^i$  for each  $i \geq 0$ , the automaton is in the same state  $q_j$ . On applying  $z$ , it reaches  $q_m$ , a final state. Hence,  $xy^i z \in L$ . As every state in  $Q$  is obtained by applying an input symbol,  $y \neq \Lambda$  (null).



- Let  $M = (Q, \Sigma, c, q_1, F)$  be a DFA recognizing  $L$  and  $p$  be the number of states of  $M$ . Let  $w = s_1 s_2 \dots s_n$  be a string in  $L$  with length  $n$ , where  $n \geq p$ . Let  $r_1, \dots, r_{n+1}$  be the sequence of states  $M$  enters when processing  $s$ .  $r_{i+1} = \delta(r_i, s_i)$  for  $1 \leq i \leq n$ . The sequence has length  $n+1$ , which is at least  $p+1$ .
- Among the first  $p+1$  elements in the sequence, two must be the same state, via the pigeonhole principle. The first is called  $r_j$ , and the second is  $r_l$ . Because  $r_l$  occurs among the first  $p+1$  places in a sequence starting at  $r_1$ , we have  $l \leq p+1$ .
-

Now let  $x = s_1 \dots s_{i-1}$ ,  $y = s_i \dots s_{l-1}$ , and  $z = s_l \dots s_n$ .  
As  $x$  takes  $M$  from  $r_1$  to  $r_i$ ,  $y$  takes  $M$  from  $r_i$  to  $r_l$ ,  
and  $z$  takes  $M$  from  $r_l$  to  $r_{n+1}$ , which is an accept  
state,  $M$  must accept  $xy^iz$  for  $i \geq 0$ .

We know  $i \neq l$ , so  $|y| > 0$ ; and  $l \leq p + 1$ , so  
 $|xy| \leq p$ .

Thus, we have satisfied all conditions of the  
pumping lemma.

- $B = \{0^n 1^n \mid n \geq 0\}$  Is this Language a Regular Language?
- Assume B is Regular, then Pumping Lemma must hold. p is the pumping length given by the PL.
- Choose s to be  $0^p 1^p$ . Because  $s \in B$  and  $|s| \geq p$ , PL guarantees s can be split into 3 pieces,  $s = xyz$ , where for any  $i \geq 0$ ,  $xy^i z \in B$ . Consider 3 cases:
  - 1. y is only 0s.  $xyyz$  has more 0s than 1s, thus a contradiction via condition 1 of PL.
  - 2. y is only 1s. Also a contradiction.
  - 3. y is both 0s and 1s.  $xyyz$  may have same number of 1s and 0s, but will be out of order, with some 1s before 0s, also a contradiction. Contradiction is unavoidable, thus B is not Regular.

# Prove that the language $\{a^k b^k \mid k \geq 0\}$ is not regular.

Prove that  $L = \{a^k b^k \mid k \geq 0\}$  is not regular.

## Step 1:

Suppose  $L$  is regular & is accepted by a FA having  $n$  states.

## Step 2:

- ♦ Let  $w = a^k b^k$  where  $k > n$
- ♦  $w \in L$
- ♦ We can write  $w = xyz$  where
  - $|xy| \leq n$
  - $|y| > 0$
- ♦ Since  $k > n$ , we have
  - $x = a^p$
  - $y = a^q$
  - $z = a^r b^n$
  - where  $p + q + r = n$  &  $q \neq 0$

## Step 3:

- ♦ Let  $i = 2$
- ♦  $w' = xy^i z$  where
  - $w' = a^p a^{2q} a^r b^n$
  - $w' = a^{p+2q+r} b^n$
  - $w' = a^{n+q} b^n$
  - Since  $q \neq 0$ ,  $w'$  has more  $a$ 's than  $b$ 's
- ♦ Hence  $w' \notin L$
- ♦ This is a contradiction, so  $L$  is not Regular

- Prove that  $L = \{a^i b^i \mid i \geq 0\}$  is not regular.
- **Solution –**
- At first, we assume that  $L$  is regular and  $n$  is the number of states.
- Let  $w = a^n b^n$ . Thus  $|w| = 2n \geq n$ .
- By pumping lemma, let  $w = xyz$ , where  $|xy| \leq n$ .
- Let  $x = a^p$ ,  $y = a^q$ , and  $z = a^r b^n$ , where  $p + q + r = n$ ,  $p \neq 0$ ,  $q \neq 0$ ,  $r \neq 0$ . Thus  $|y| \neq 0$ .
- Let  $k = 2$ . Then  $xy^2z = a^p a^{2q} a^r b^n$ .
- Number of  $a$ s =  $(p + 2q + r) = (p + q + r) + q = n + q$
- Hence,  $xy^2z = a^{n+q} b^n$ . Since  $q \neq 0$ ,  $xy^2z$  is not of the form  $a^n b^n$ .
- Thus,  $xy^2z$  is not in  $L$ . Hence  $L$  is not regular.

# Closure Property of Regular Set

- (i) Union,
- (ii) Concatenation
- (iii) Closure (iteration)
- (iv) Transpose
- (v) Intersection
- (vi) Complementation.

Class of regular sets is closed under **union, concatenation and closure.**

# Questions for practice on RE

- 1. find regular expression  $L \in \{a,b\}^*$ 
  - ▣ Set of all strings ending in b:  $(a+b)^* b$
  - ▣ Set of all strings ending in ba:  $(a+b)^* ba$
  - ▣ Set of all strings ending neither in b nor in ba:  $(a+b)^* aa+a+ \epsilon$
  - ▣ Set of all strings ending neither in ab nor ba:  
 $\epsilon+a+b+(a+b)^*(aa+bb)$
- 2.

# TAFL-Course Outcomes

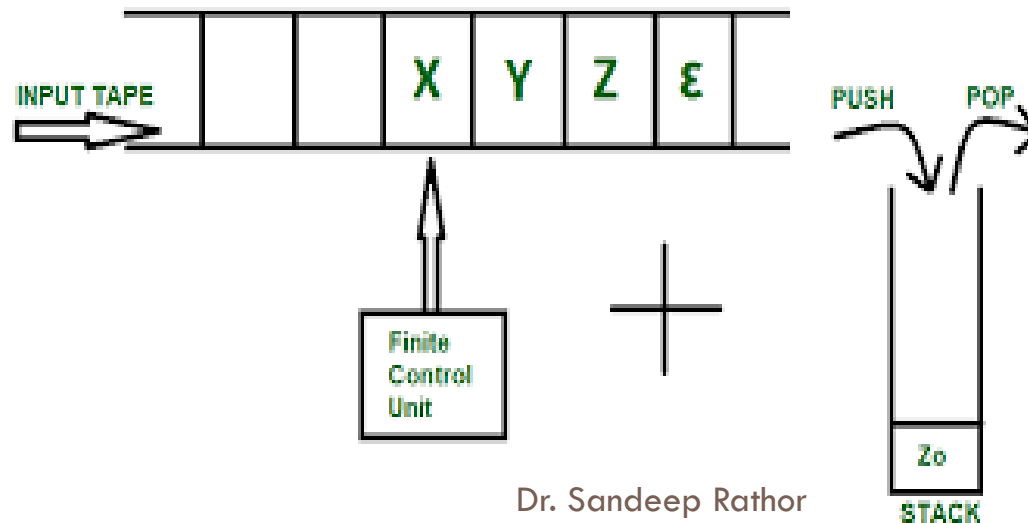
|            |  |                   |
|------------|--|-------------------|
| <b>CO1</b> | <b>Understand and interpret Context Free languages, Expression and Grammars.</b>   | <b>U</b>          |
| <b>CO2</b> | <b>Understand the limitations of each model of computation.</b>  | <b>U</b>          |
| <b>CO3</b> | <b>Apply model of computation to different problems.</b>   | <b>Ap</b>         |
| <b>CO4</b> | <b>Develop analytical thinking and intuition for problem solving situations in related areas of theory of computation.</b> | <b>Ap</b>         |
| <b>CO5</b> | <b>Understand the limitations of computation, i.e. the insolvability of problems.</b>                                      | <b>U, Ap</b>      |
| <b>CO6</b> | <b>Compare, understand and analyze different languages, grammars, Automata and Machines.</b>                               | <b>U, Analyze</b> |
| <b>CO7</b> | <b>convert grammar/regular expression into respective automata and vice-versa.</b>   | <b>Ap</b>         |



# TAFL- Credits: 4:0:0

|     | Course Outcome  | POs/<br>PSOs                 | Cognitive Level | KC           | Class Sessions |
|-----|---|------------------------------|-----------------|--------------|----------------|
| CO1 | Understand and interpret Context Free languages, Expression and Grammars.   | PO1,<br>PO10,<br>PSO1        | U               | F, C         | 3              |
| CO2 | Understand the limitations of each model of computation.  | PO1,<br>PO10,<br>PSO1        | U               | C            | 9              |
| CO3 | Apply model of computation to different problems.   | PO1,<br>PSO1                 | Ap              | C, P         | 4              |
| CO4 | Develop analytical thinking and intuition for problem solving situations in related areas of theory of computation. | PO3,<br>PO4,<br>PO5,<br>PSO1 | Ap              | C, P,<br>MC  | 10             |
| CO5 | Understand the limitations of computation, i.e. the insolvability of problems.                                      | PO3,<br>PO4,<br>PO5,<br>PSO1 | U, Ap           | C, P         | 8              |
| CO6 | Compare, understand and analyze different languages, grammars, Automata and Machines.                               | PO3,<br>PO4,<br>PO5,         | U, Analyze      | C, P,<br>C&S | 6              |

# PDA (Push Down Automata)



Dr. Sandeep Rathor

A pushdown automaton (PDA) consists of seven-tuple:

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

$Q$  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,

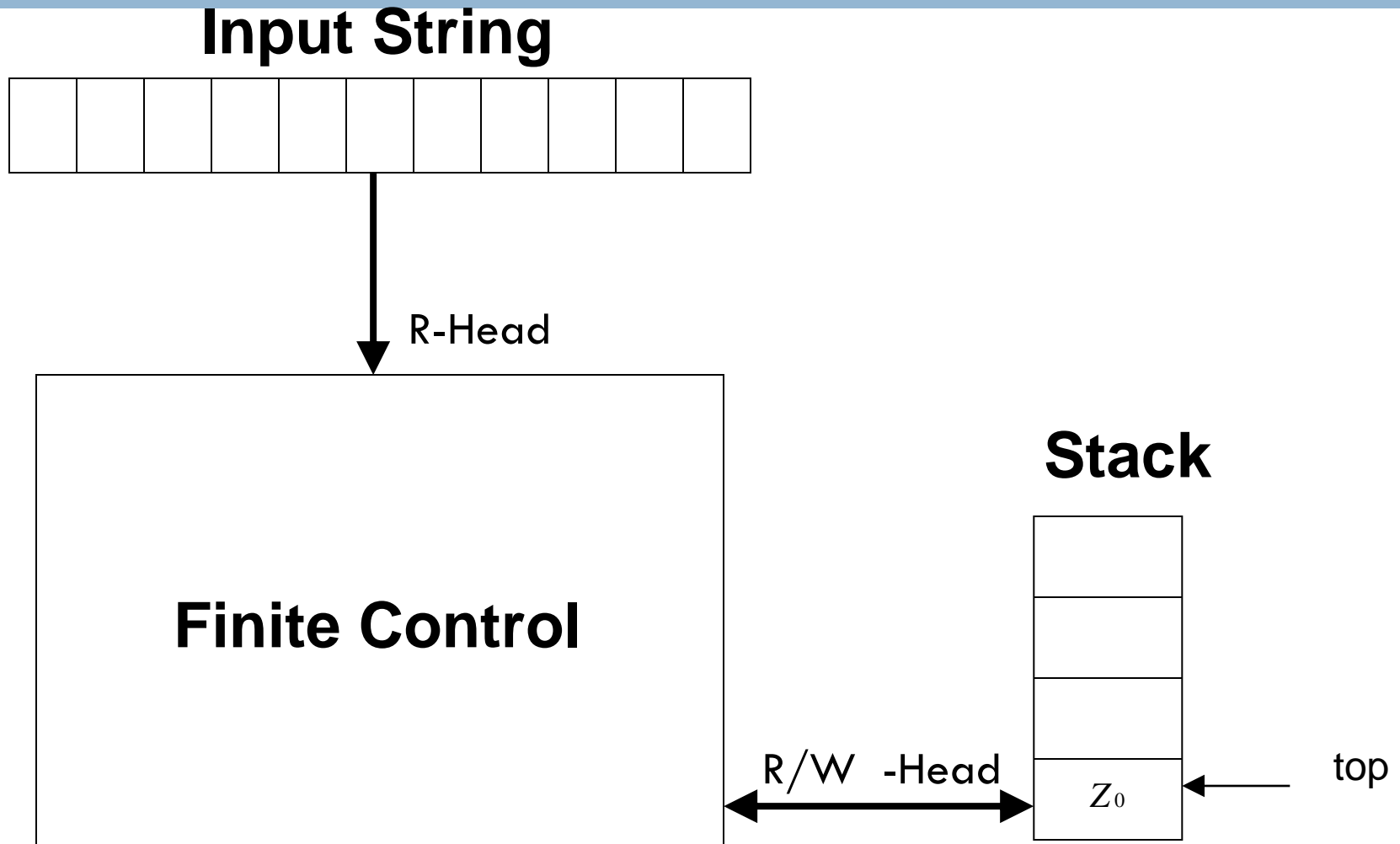
$F$  A set of final/accepting states, which is a subset of

$\delta$  A transition function, where

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

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

# Model of PDA

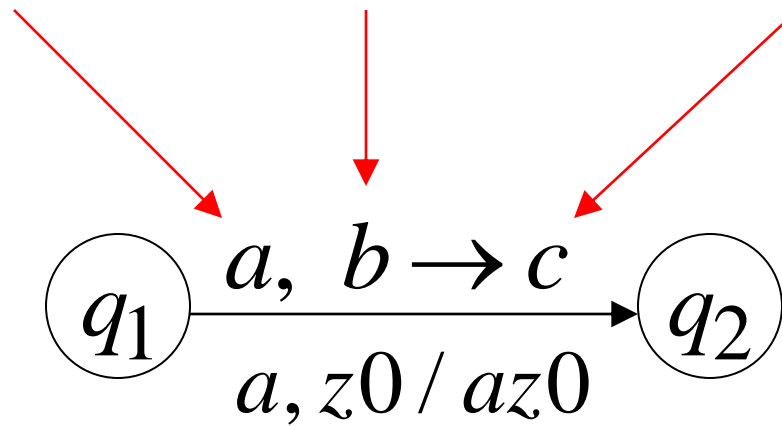


# The States

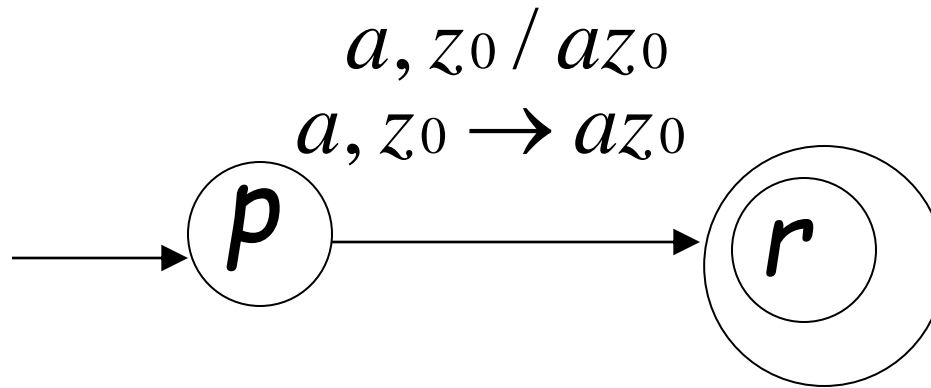
Input  
symbol

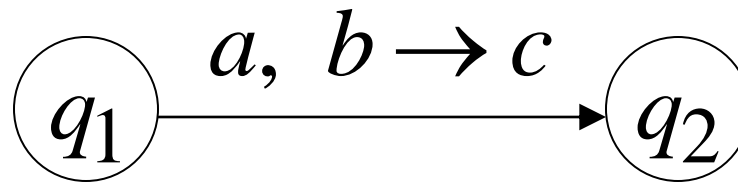
Pop  
symbol

Push  
symbol

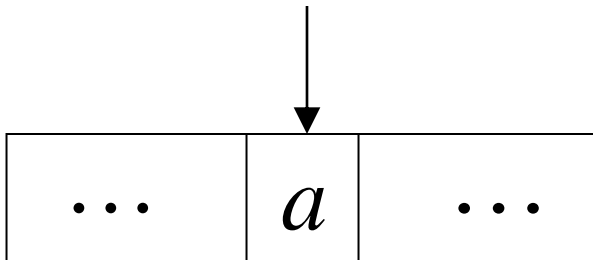


# State Representation & Execution

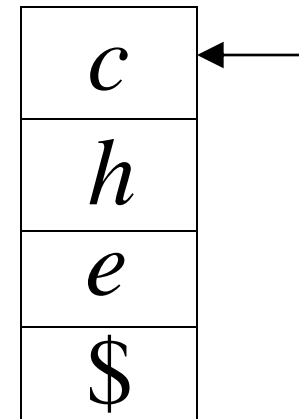
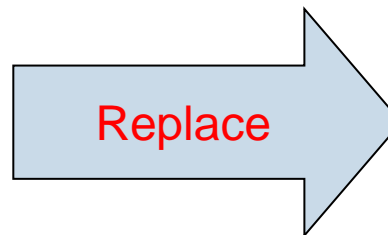
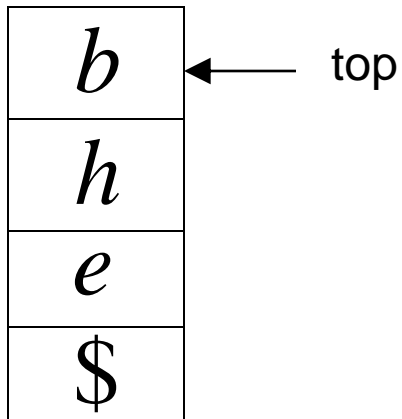


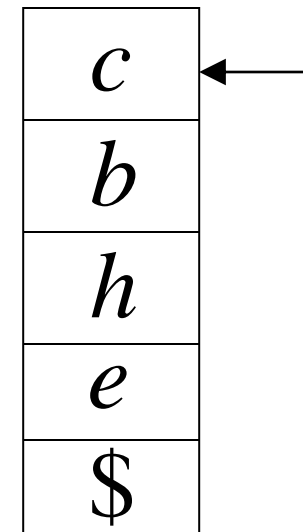
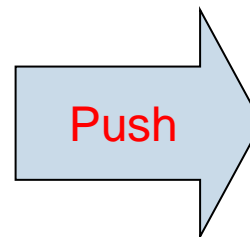
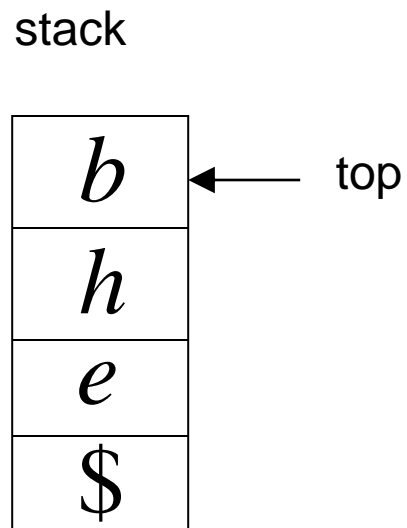
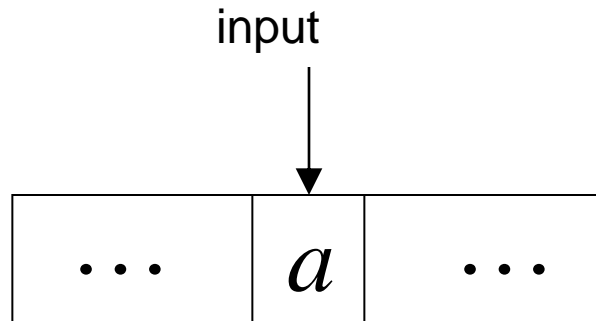
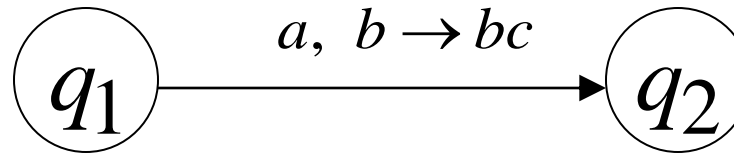


input

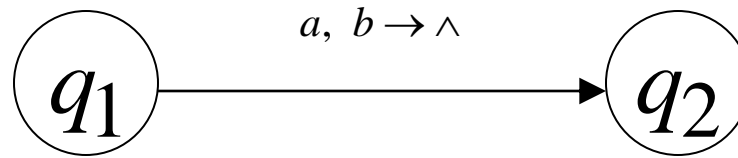


stack

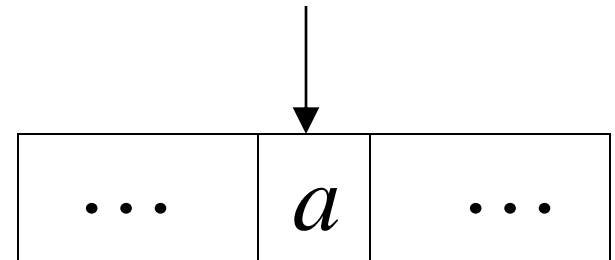




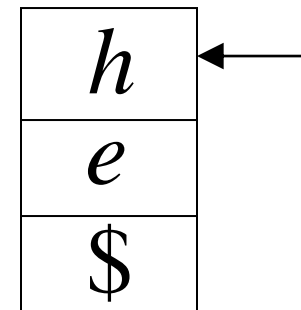
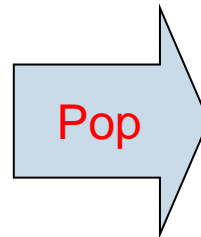
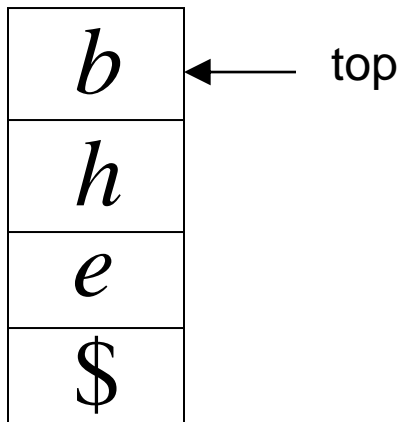




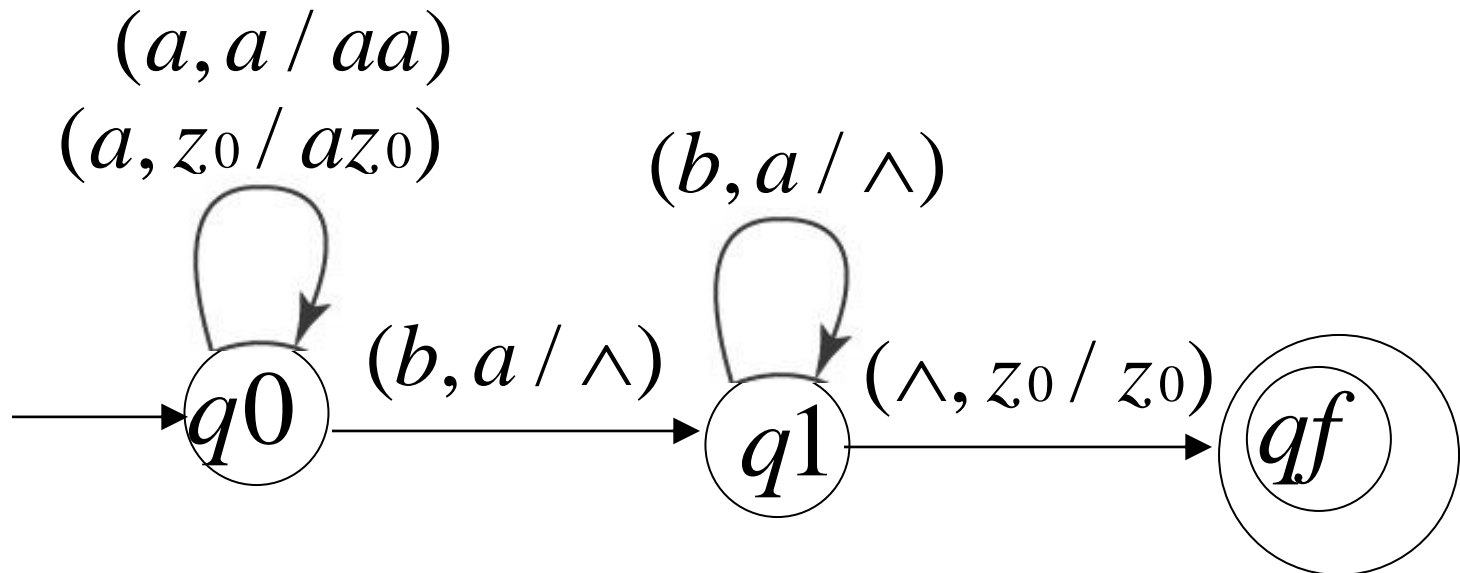
input



stack



# PDA for $L = \{a^n b^n : n \geq 1\}$



$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$

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

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

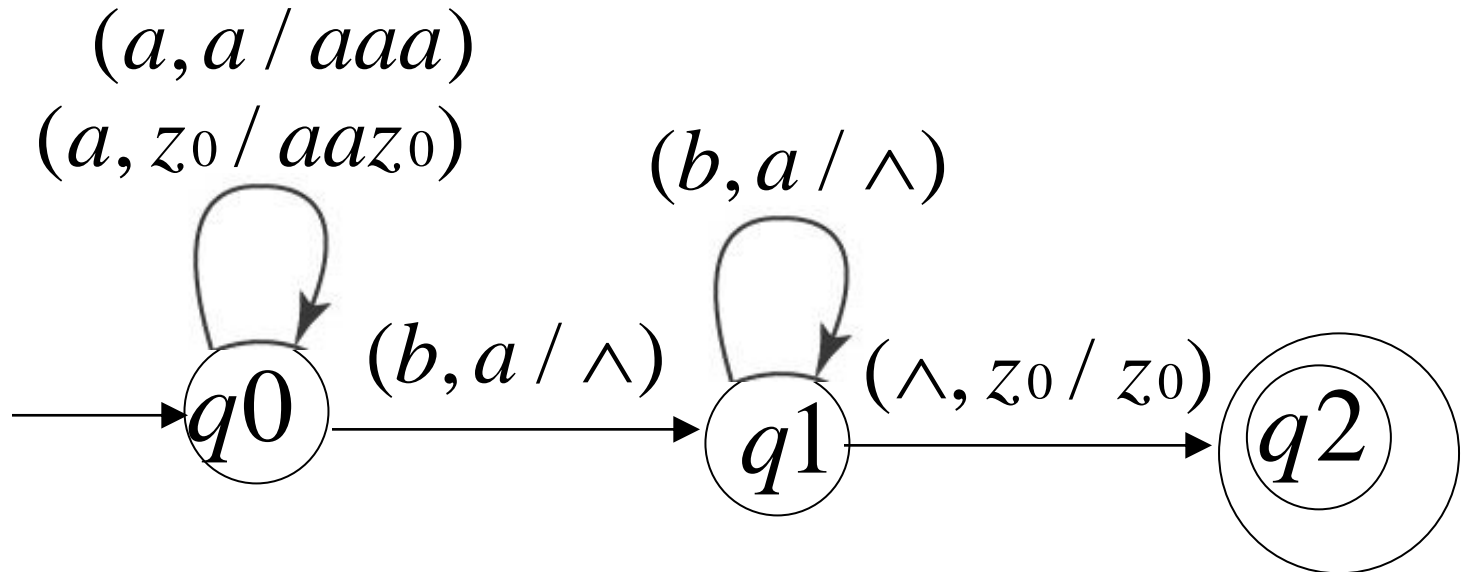
$$\delta(q_1, b, a) = (q_1, \epsilon)$$

$$\delta(q_1, \epsilon, Z_0) = (q_f, Z_0) \quad // \text{ By Final State}$$

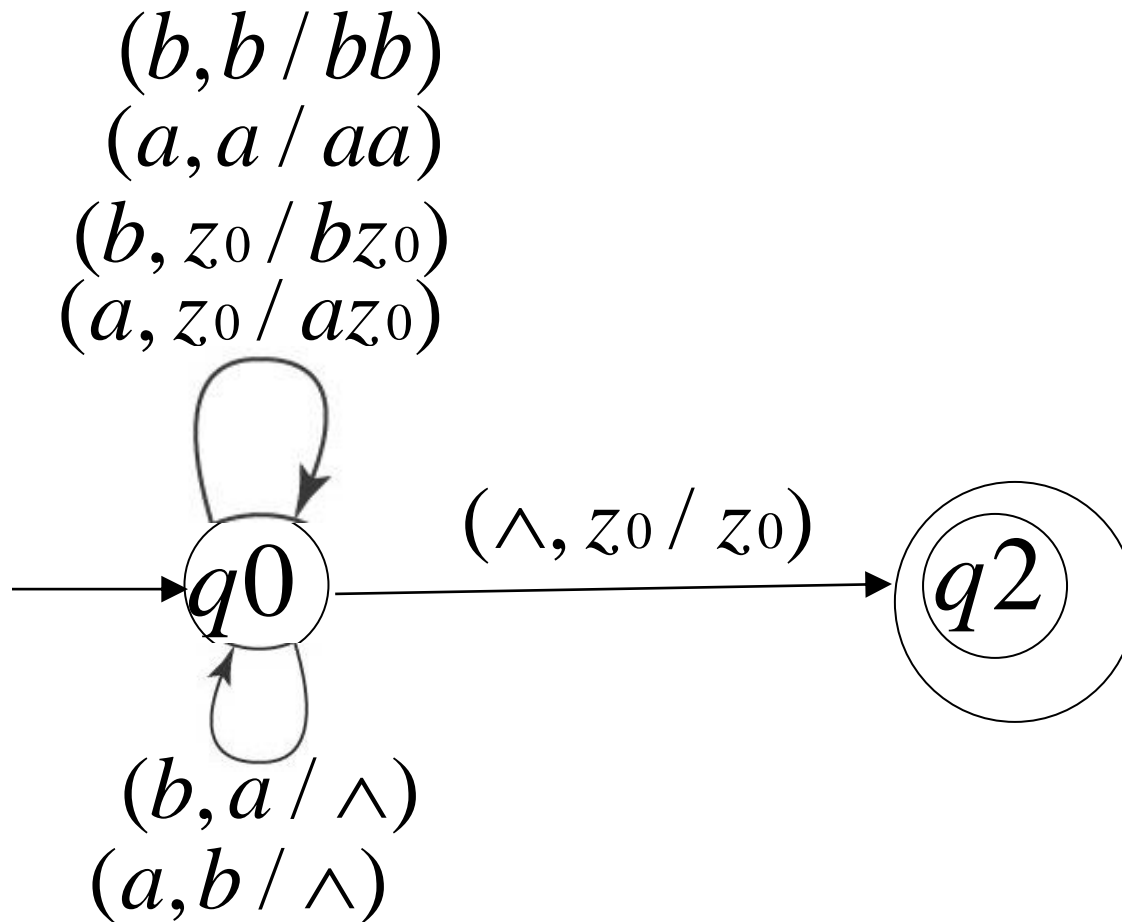
OR

$$\delta(q_1, \epsilon, Z_0) = (q_1, \epsilon) // \text{ by empty stack}$$

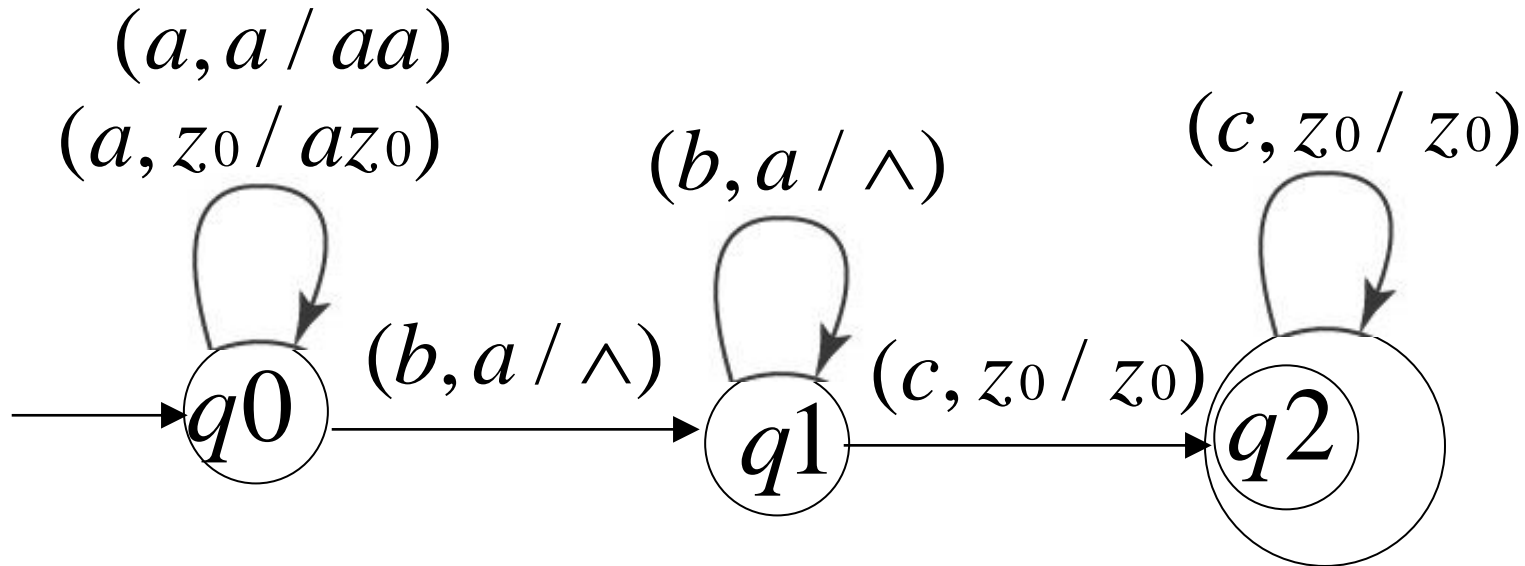
# PDA for $L = \{ a^n b^{2n} : n \geq 1 \}$



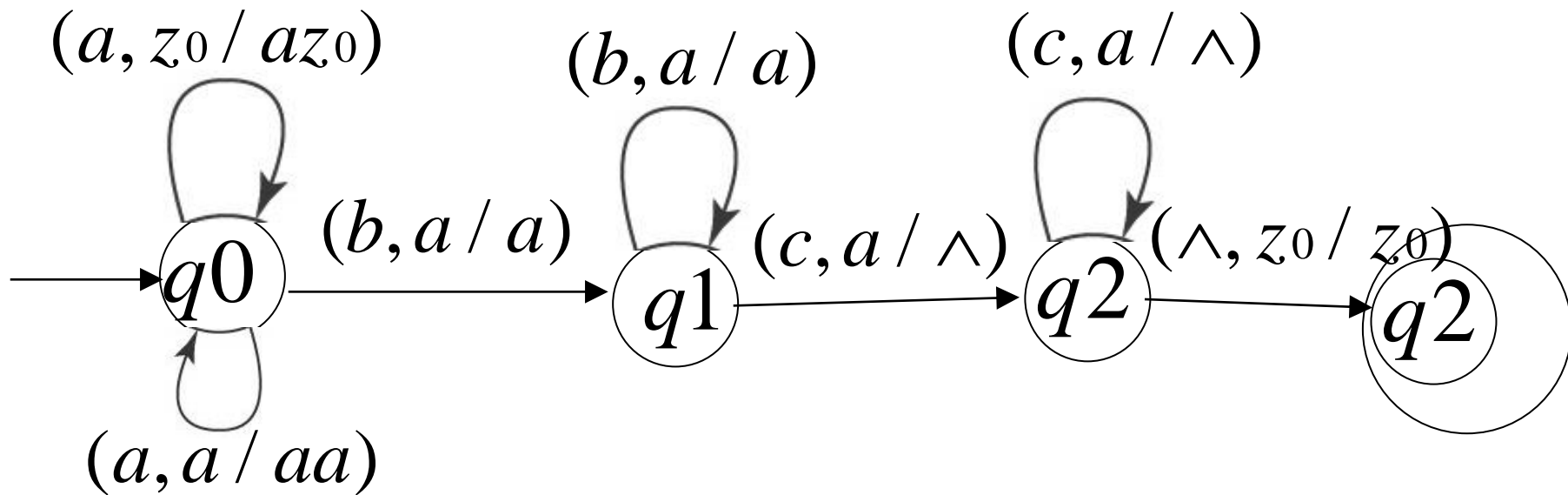
# PDA for $L = \{w \mid n_a(w) = n_b(w)\}$



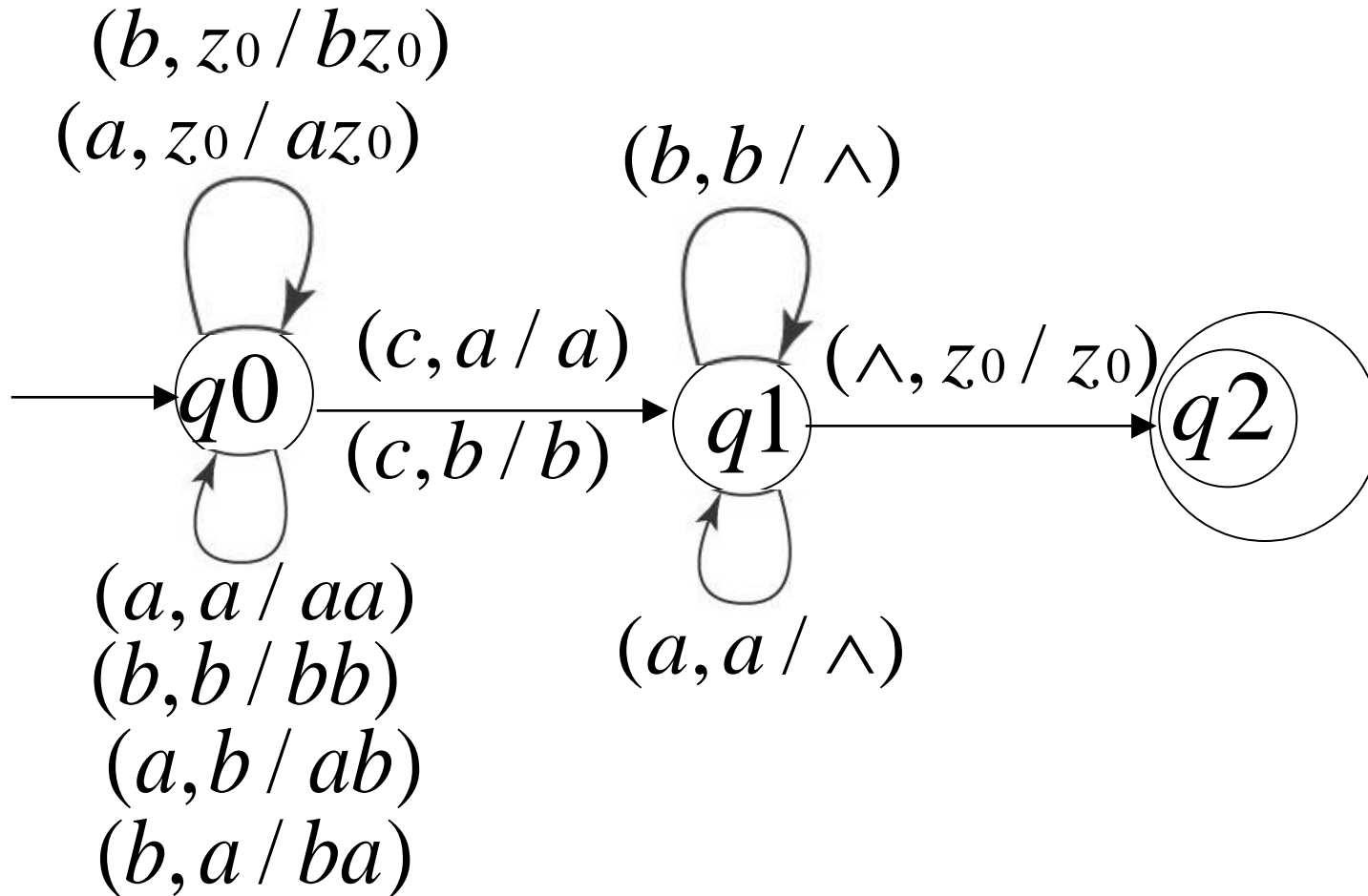
# PDA for $L = \{ a^n b^n c^m : n, m \geq 1 \}$



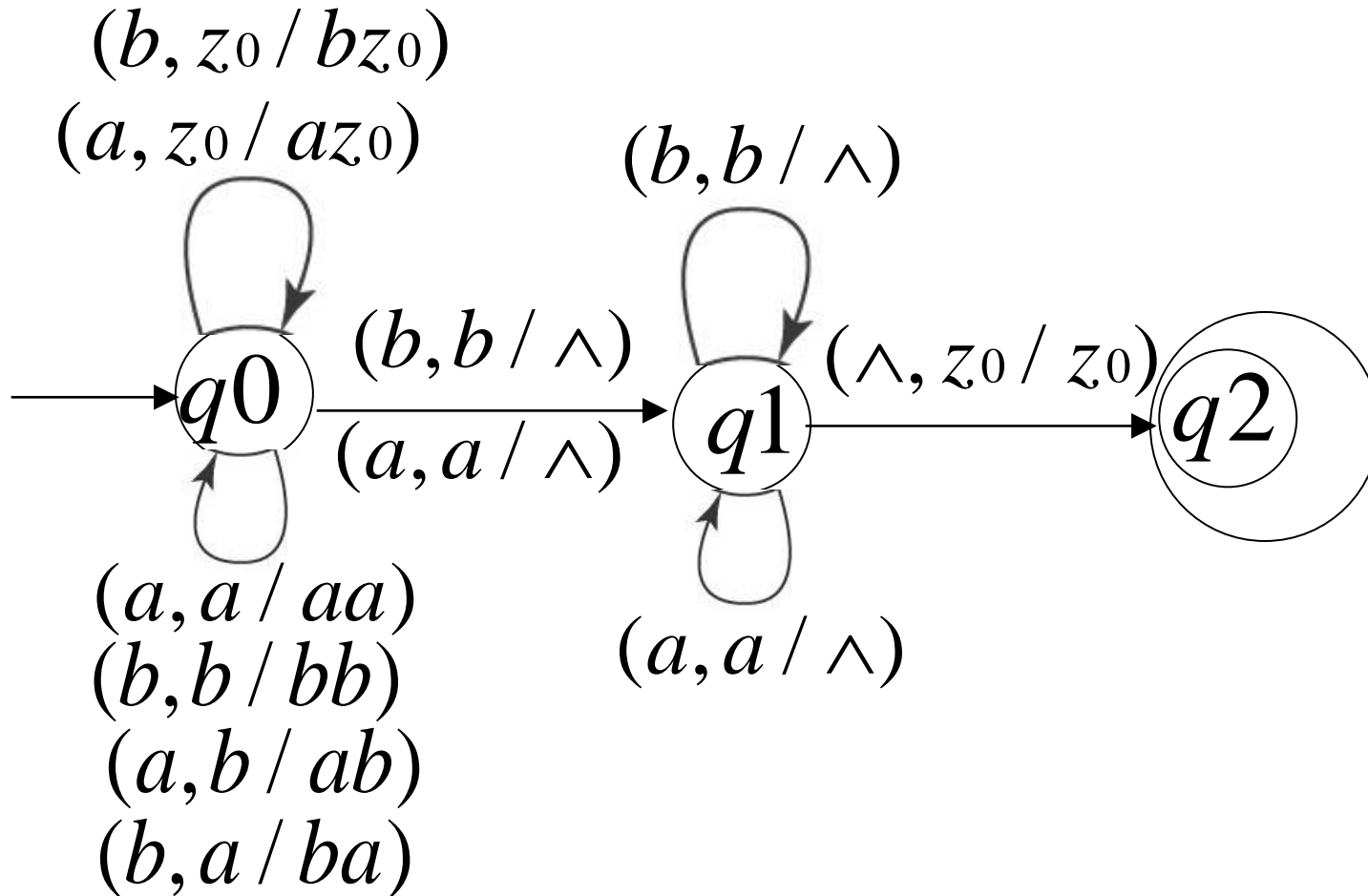
# PDA for $L = \{ a^n b^m c^n : n, m \geq 1 \}$



# PDA for $L = \{wcw^R : w \in (a,b)^+\}$

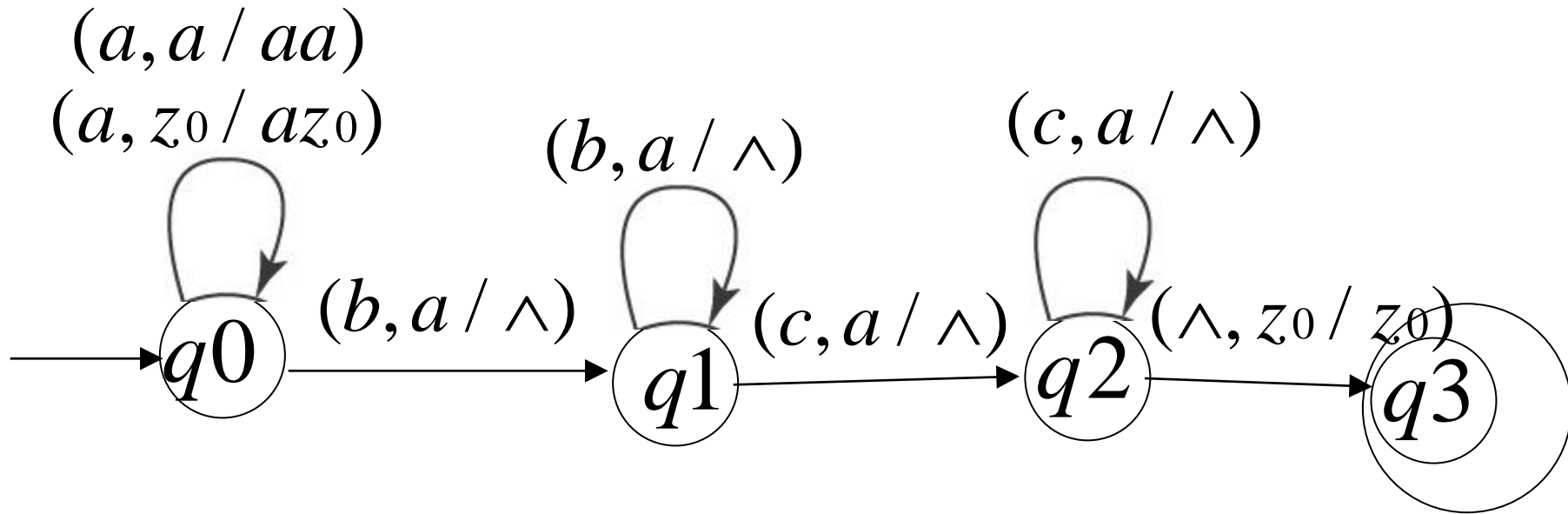


# PDA for $L = \{ ww^R : w \in (a,b)^+ \}$

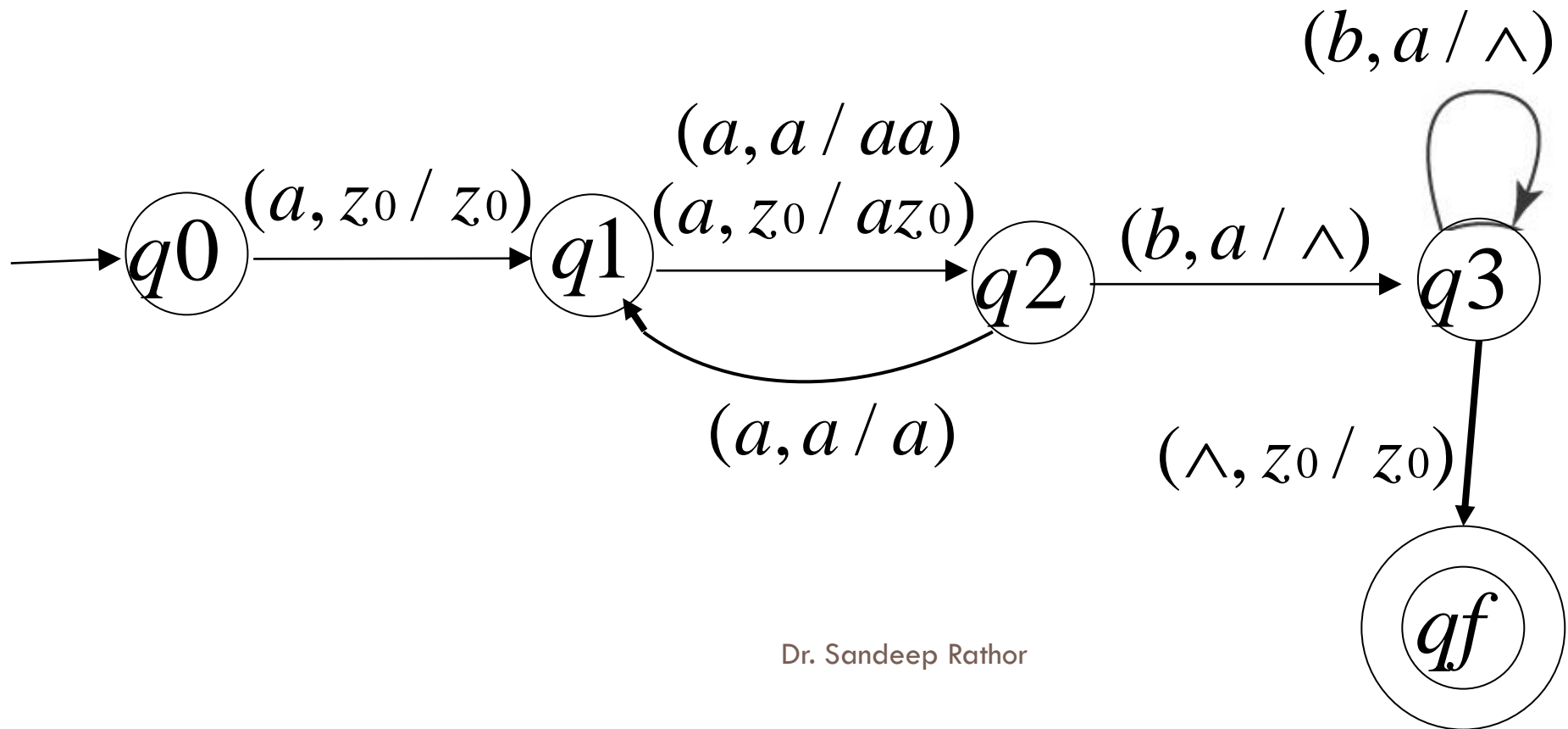




**PDA for  $L = \{ a^{m+n} b^m c^n : n, m \geq 1 \}$**



# PDA for $L = \{ a^{2n}b^n : n \geq 1 \}$



# Applications of PDA

Online Transaction process system

Tower of Hanoi (Recursive Solution)

Timed Automata Model

Deterministic Top Down Parsing LL Grammar

Context free Language

Predictive Bottom up Parsing LR Grammar

For implementation of stack applications.

For evaluating the arithmetic expressions.

# Applications of TM

- For solving any recursively enumerable problem.
- For understanding complexity theory.
- For implementation of neural networks.
- For implementation of Robotics Applications.
- For implementation of artificial intelligence.

# Q & A

1. Give a CFG for language  $L = \{x \in \{0,1\}^* \mid x \text{ start and ends with different symbol}\}$

$$S \rightarrow 0A1 / 1A0$$

$$A \rightarrow 0A / 1A / \wedge$$

2. Which of these not in GNF:

$$S \rightarrow AS / SBB / a$$

$$A \rightarrow bAA / b$$

$$B \rightarrow ab / Ba$$

*Ans*

$$S \rightarrow AS / SBB / a$$

$$B \rightarrow ab / Ba$$

3. Prove that following grammar is ambiguous

$$S \rightarrow aS / aSbS / c$$

4. Convert following grammar into CNF:

$$S \rightarrow bA / aB$$

$$A \rightarrow bAA / aS / a$$

$$B \rightarrow aBB / bS / b$$

5. Convert following grammar to PDA:

$$S \rightarrow aAA$$

$$A \rightarrow aS / bS / a$$

6. Eliminate Null and Unit production:

$$S \rightarrow aXbX$$

$$X \rightarrow aY / bY / \wedge$$

$$Y \rightarrow X / c$$

*After Null*

$$S \rightarrow aXbX / abX / aXb / ab$$

$$X \rightarrow aY / bY / a / b$$

$$Y \rightarrow X / c$$

*After Unit*

$$S \rightarrow aXbX / abX / aXb / ab$$

$$X \rightarrow aY / bY / a / b$$

Dr. Sandeep Rathi

$$Y \rightarrow aY / bY / a / b / c$$

# CFG to PDA



- ◆ The PDA simulates the **left-sentential forms** that  $G$  uses to generate any string  $w$
- ◆ Let  $G = (V, \Sigma, P, S)$
- ◆ Construct PDA  $N$  that **accepts  $L(G)$  by empty stack**
- ◆  **$N = (\{q\}, \Sigma, V \cup \Sigma, \delta, q, S)$**
- ◆ Transitions are defined as

- For each variable  $A$

$$\delta(q, \epsilon, A) = \{(q, \beta) \mid A \rightarrow \beta \text{ is a prod}^n \text{ in } G\}$$

- For each terminal  $a$

$$\delta(q, a, a) = \{(q, \epsilon)\}$$

Find a PDA equivalent to the grammar  **$S \rightarrow aSbb \mid a$**   
Generate the string aabb and also simulate the PDA for it

◆  **$N = (\{q\}, \Sigma, V \cup \Sigma, \delta, q, S)$**

Where  $\Sigma = \{a, b\}$ ,  $V = \{S\}$  &  $\delta$  is  
as following

1.  $\delta(q, \varepsilon, S) = \{(q, aSbb), (q, a)\}$
2.  $\delta(q, a, a) = \{(q, \varepsilon)\}$
3.  $\delta(q, b, b) = \{(q, \varepsilon)\}$

**$S \Rightarrow aSbb \Rightarrow aabb$**

(q, aabb, S)  
├ (q, aabb, aSbb)  
├ (q, abb, Sbb)  
├ (q, abb, abb)  
├ (q, bb, bb)  
├ (q, b, b)  
└ (q,  $\varepsilon$ ,  $\varepsilon$ )

Stack Empty – string  
accepted

## Construction of PDA by a CFG

$$\delta(q, \epsilon, A) = \{(q, \alpha) / A \rightarrow \alpha \text{ in } P\}$$

$$\delta(q, t, t) = \{(q, \epsilon)\} \text{ for every } t \text{ in } \Sigma$$

1. Construct PDA equivalent to following CFG:

$$S \rightarrow 0/1/1A$$

$$A \rightarrow 0/1S$$

Ans:

$$\delta(q, \epsilon, S) = \{(q, 0), (q, 1), (q, 1A)\}$$

$$\delta(q, \epsilon, A) = \{(q, 0), (q, 1S)\}$$

$$\delta(q, 0, 0) = \{(q, \epsilon)\}$$

$$\delta(q, 1, 1) = \{(q, \epsilon)\}$$

2. Convert following CFG to PDA

$$S \rightarrow aAA$$

$$A \rightarrow aS/bS/a$$



Find a PDA equivalent to the following grammar that generates simple expressions like  $a * (a + b010)$

$$E \rightarrow I \mid E * E \mid E + E \mid (E)$$

$$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

Let  $N$  be a PDA that accepts the language generated

by the given CFG

$$N = (\{q\}, \Sigma, V \cup \Sigma, \delta, q, S)$$

$$\Sigma = \{a, b, 0, 1, (, ), *, +\}$$

$$V = \{E, I\}$$

1.  $\delta(q, \varepsilon, E) = \{(q, I), (q, E^*E), (q, E+E), (q, (E))\}$
2.  $\delta(q, \varepsilon, I) = \{(q, a), (q, b), (q, Ia), (q, Ib), (q, I0), (q, I1)\}$
3.  $\delta(q, a, a) = \{(q, \varepsilon)\}$
4.  $\delta(q, b, b) = \{(q, \varepsilon)\}$
5.  $\delta(q, 0, 0) = \{(q, \varepsilon)\}$
6.  $\delta(q, 1, 1) = \{(q, \varepsilon)\}$
7.  $\delta(q, (, () = \{(q, \varepsilon)\}$
8.  $\delta(q, ), )) = \{(q, \varepsilon)\}$
9.  $\delta(q, *, *) = \{(q, \varepsilon)\}$
10.  $\delta(q, +, +) = \{(q, \varepsilon)\}$

# PDA to CFG

We construct grammar  $G$  as follows.

$$G = (V, \Sigma, P, S)$$

$$\boxed{\text{V}} \quad V = \{S\} \cup \{q, z, q' \mid q, q' \in Q, z \in \Gamma\}$$

The productions in  $P$  are induced by the moves of PDA as follows

$\boxed{\text{Rule 1:}}$

The  $S$  productions are given by

$$S \rightarrow [q_0, Z_0, q] \quad \forall q \in Q$$

## ▣ Rule 2:

Each move erasing a pushdown symbol given by  $(q', \varepsilon) \in \delta(q, a, z)$  induces the prod<sup>n</sup>

$$[q, z, q'] \rightarrow a$$

## ▣ Rule 3:

Each move not erasing a pushdown symbol given by  $(q_1, z_1 z_2 \dots z_m) \in \delta(q, a, z)$  induces many prod<sup>n</sup>s of the following form where  $q', q_2, \dots, q_m$  can be any state in  $Q$

$$[q, z, q'] \rightarrow a [q_1, z_1, q_2] [q_2, z_2, q_3] \dots [q_m, z_m, q']$$

Construct a context-free grammar  $G$  which accepts  $L(N)$ , where

$$N = (\{q_0, q_1\}, \{a, b\}, \{Z_0, Z\}, \delta, q_0, Z_0, \varphi)$$

$\delta$  is given by

1.  $\delta(q_0, b, Z_0) = \{(q_0, ZZ_0)\}$
2.  $\delta(q_0, \varepsilon, Z_0) = \{(q_0, \varepsilon)\}$
3.  $\delta(q_0, b, Z) = \{(q_0, ZZ)\}$
4.  $\delta(q_0, a, Z) = \{(q_1, Z)\}$
5.  $\delta(q_1, b, Z) = \{(q_1, \varepsilon)\}$
6.  $\delta(q_1, a, Z_0) = \{(q_0, Z_0)\}$

Let  $G = \{V, \{a, b\}, P, S\}$

$$\boxed{\square} \quad V = \{S\} \cup \{q, z, q'\} \mid q, q' \in Q, z \in \Gamma\}$$

$$V = \{S\} \cup \{[q_0, Z_0, q_0], [q_0, Z_0, q_1], \\ [q_0, Z_1, q_0], [q_0, Z_1, q_1], \\ [q_1, Z_0, q_0], [q_1, Z_0, q_1], \\ [q_1, Z_1, q_0], [q_1, Z_1, q_1],$$

The productions in  $P$  are induced by the moves of PDA

▣ **Rule 1:**  $S \rightarrow [q_0, Z_0, q] \forall q \in Q$

**1.**  $S \rightarrow [q_0, Z_0, q_0]$

**2.**  $S \rightarrow [q_0, Z_0, q_1]$

▣ **Rule 2:**  $\delta(\mathbf{q}, \mathbf{a}, \mathbf{z}) = (\mathbf{q}', \varepsilon)$  induces  $[\mathbf{q}, \mathbf{z}, \mathbf{q}'] \rightarrow \mathbf{a}$

- $\delta(q_0, \varepsilon, Z_0) = \{(q_0, \varepsilon)\}$

**3.  $[q_0, Z_0, q_0] \rightarrow \varepsilon$**

- $\delta(q_1, b, Z) = \{(q_1, \varepsilon)\}$

**4.  $[q_1, Z, q_1] \rightarrow b$**



▣ **Rule 3:**  $\delta(q, a, z) = (q_1, z_1 z_2 \dots z_m)$

$$[q, z, q'] \rightarrow a [q_1, z_1, q_2] [q_2, z_2, q_3] \dots [q_m, z_m, q']$$

■  $\delta(q_0, b, Z_0) = \{(q_0, ZZ_0)\}$

$$[q_0, Z, \blacklozenge] \rightarrow b [q_0, Z, \blacklozenge] [\blacklozenge, Z_0, \blacklozenge]$$

$$5. [q_0, Z_0, q_0] \rightarrow b [q_0, Z, q_0] [q_0, Z_0, q_0]$$

$$6. [q_0, Z_0, q_0] \rightarrow b [q_0, Z, q_1] [q_1, Z_0, q_0]$$

$$7. [q_0, Z_0, q_1] \rightarrow b [q_0, Z, q_0] [q_0, Z_0, q_1]$$

$$8. [q_0, Z_0, q_1] \rightarrow b [q_0, Z, q_1] [q_1, Z_0, q_1]$$

- $\delta(q_0, b, Z) = \{(q_0, ZZ)\}$

$$[q_0, Z, \blacklozenge] \rightarrow b [q_0, Z, \blacklozenge] [\blacklozenge, Z, \blacklozenge]$$

$$9. [q_0, Z, q_0] \rightarrow b [q_0, Z, q_0] [q_0, Z, q_0]$$

$$10. [q_0, Z, q_0] \rightarrow b [q_0, Z, q_1] [q_1, Z, q_0]$$

$$11. [q_0, Z, q_1] \rightarrow b [q_0, Z, q_0] [q_0, Z, q_1]$$

$$12. [q_0, Z, q_1] \rightarrow b [q_0, Z, q_1] [q_1, Z, q_1]$$

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

$$[q_0, Z, \blacklozenge] \rightarrow a [q_1, Z, \blacklozenge]$$

$$13. [q_0, Z, q_0] \rightarrow a [q_1, Z, q_0]$$

$$14. [q_0, Z, q_1] \rightarrow a [q_1, Z, q_1]$$

- $\delta(q_1, a, Z_0) = \{(q_0, Z_0)\}$

$$[q_1, Z_0, \blacklozenge] \rightarrow a [q_0, Z_0, \blacklozenge]$$

$$15. [q_1, Z_0, q_0] \rightarrow a [q_0, Z_0, q_0]$$

$$16. [q_1, Z_0, q_1] \rightarrow a [q_0, Z_0, q_1]$$

# Closure Properties of Languages

| <i>Property</i>                   | <i>Regular</i> | <i>CFL</i> | <i>DCFL</i> | <i>CSL</i> | <i>Recursive</i> | <i>RE</i> |
|-----------------------------------|----------------|------------|-------------|------------|------------------|-----------|
| Union                             | Yes            | Yes        | No          | Yes        | Yes              | Yes       |
| Intersection                      | Yes            | No         | No          | Yes        | Yes              | Yes       |
| Set Difference                    | Yes            | No         | No          | Yes        | Yes              | No        |
| Complementation                   | Yes            | No         | Yes         | Yes        | Yes              | No        |
| Intersection with a regular lang. | Yes            | Yes        | Yes         | Yes        | Yes              | Yes       |
| Concatenation                     | Yes            | Yes        | No          | Yes        | Yes              | Yes       |
| Kleen Closure                     | Yes            | Yes        | No          | Yes        | Yes              | Yes       |
| Kleen Plus                        | Yes            | Yes        | No          | Yes        | Yes              | Yes       |
| Reversal                          | Yes            | Yes        | No          | Yes        | Yes              | Yes       |
| Homomorphism                      | Yes            | Yes        | No          | No         | No               | Yes       |
| $\epsilon$ -free Homomorphism     | Yes            | Yes        | No          | Yes        | Yes              | Yes       |
| Inverse Homomorphism              | Yes            | Yes        | Yes         | Yes        | Yes              | Yes       |
| Substitution                      | Yes            | Yes        | No          | No         | No               | Yes       |
| $\epsilon$ -free Substitution     | Yes            | Yes        | No          | Yes        | Yes              | Yes       |

# Decision Properties

## Emptiness

No unreachable states

Free from final state

If having final state then **"non-empty"**

If having cycle/loop then **"in-finite"**

## Finiteness

No unreachable & dead states

accepts only non empty strings

Only bounded length

No loops/cycles

## Membership

Verifies arbitrary string accepted by FA or NOT

Whether a member of a Language or not

i.e. is string  $w$  in regular language  $L$ ?

## Equality

Accepts same language

Recognizes same token