



# MIT-WPU

## SY B.Tech(CSE-AIDS)

# Theory of Computation

## (CET2008B)

# Course Objective & Course Outcomes

- **Course Objectives:**

1. **Knowledge:**

- (i) To study equivalence of regular expressions and Finite Automata.

2. **Skill:**

- (i) To demonstrate the concepts of Automata theory from the perspective of formal languages.

3. **Attitude:**

- I. To analyze context-free grammar and Pushdown Automata.
- II. To design a Turing machine and to understand computability and complexity theory.

# Course Objective & Course Outcomes

- **Course Outcomes:**
  1. To construct Finite Automata to solve problems in computing.
  2. To build regular expressions and understand regular language.
  3. To construct context-free grammar and Push Down Automata.
  4. To design computational models and classify the problems of decidability.

# Text Books & Reference Books

- **Text Books**

- Vivek Kulkarni, Theory of computation, Oxford University Press, ISBN-13: 978-0-19-808458-7.
- K.L.P Mishra, N. Chandrasekaran, Theory of Computer Science (Automata, Languages and Computation), Prentice Hall India, 2nd Edition.

- **Reference Books**

1. John C. Martin, Introduction to Language and Theory of Computation, TMH, 3<sup>rd</sup> Edition, ISBN: 978-0-07-066048-9.
2. Michael Sipser, Introduction to the Theory of Computation, CENGAGE Learning, 3<sup>rd</sup> Edition, ISBN: 13:978-81-315-2529-6.
3. Daniel Cohen, Introduction to Computer Theory, Wiley India, 2nd Edition, ISBN: 9788126513345.
4. Kavi Mahesh, Theory of Computation: A Problem Solving Approach, 1st Edition, Wiley-India, ISBN: 978-81-265-3311-4.

# Unit I

- Introduction to Formal language, Basic concepts: Symbol, Alphabet, String
- Introduction to Finite Automata, State transition graph, Transition table, Acceptance of a string, Acceptance of a Language
- Deterministic finite Automata (DFA)-Formal Definition
- Non-Deterministic finite Automata (NFA)-Formal Definition
- Non-Deterministic finite Automata (NFA) with epsilon transition
- Equivalence of NFA and DFA, Conversion from NFA to DFA, Conversion from NFA with epsilon transition to DFA
- Applications and Limitations of FA.

# Planner - Lectures

Lecture No	Topics Covered
1	Introduction to TOC, Introduction to Formal language, Basic concepts: Symbol, Alphabet, String
2	Introduction to FA, State transition graph, Transition table, Acceptance of a string, Acceptance of a Language,
3	DFA -Formal Definition, NFA-Formal Definition, NFA with epsilon transition, Equivalence of NFA and DFA
4	Conversion from NFA to DFA
5	Conversion from NFA to DFA (cont), Conversion from NFA with epsilon transition to DFA
6-8	Conversion from NFA with epsilon transition to DFA (cont)
9	Minimization of FA
10	Applications and Limitations of FA.

# About TOC

- Course is about models of computation, their power, and relationships.
- Hierarchy of models of increasing power:

-DFA < PDA < TM.

For each model, we study:

- What can be computed.
- What cannot be computed.

- Automata theory is the study of abstract computing devices or “machines”

# History of TOC

- 1930-45: Alan Turing studied Turing machine  
Describe boundary between what a computing machine could do and what it could not do
- 1940-50: Researchers studies Finite Automata
- Late 1950s: N. Chomsky studied formal “Grammars”
- 1969-70: S. Cook extended Turing’s study  
Separated problems that can be solved efficiently by computers and the problems that can be solved theoretically, but takes so much time by computers to solve



# Automata theory

- The study of **abstract 'mathematical' machines** or systems and the computational problems that can be solved using these machines.
- These abstract machines are called automata.
- Automata theory is also closely related to **formal language** theory, as the automata are often classified by the class of formal languages they are able to recognize.
- An automaton can be a finite representation of a formal language that may be an infinite set.
- Automata are used as theoretical models for computing machines, and are used for proofs about computability.

# Basic Definitions

- **Difference Between Formal Language and Natural Language**

Formal Language	Natural Language
1. All the rules are stated explicitly	1. Not necessarily rules to be stated explicitly. It is a medium of communication.
2. Considered as Symbols on papers	2. Expressions of ideas in Natural way.
3. In formal Languages we are interested in the form of strings of symbols and not meaning	3. We are interested in Meaning than only strings.

# Basic Definitions

- **Alphabet** - a finite set of symbols.

Notation:  $\Sigma$ .

E.g.: Binary alphabet  $\{0,1\}$ ,

English alphabet  $\{a,\dots,z,!,?\}$

- **String over an alphabet  $\Sigma$**  - a finite sequence of symbols from  $\Sigma$ .

E.g: 0100 is a string over the binary alphabet.

a!? is a string over the English alphabet.

- **Language over alphabet  $\Sigma$**  - a set of strings over  $\Sigma$ .

Notation:  $L$ .

E.g:

$\{0, 00, 000, \dots\}$  is an "infinite" language over the binary alphabet.

$\{a, b, c\}$  is a "finite" language over the English alphabet.

# Basic Definitions (cont...)

- **Empty string:**  $\epsilon$  or  $\varepsilon$  denotes the empty sequence of symbols.
- **Empty Language:** Empty set of strings.  
Notation:  $\Phi$
- **Regular Language-** a finite set of symbols.  
L is called regular Language if there is some automaton that accepts L.  
E.g.  $\Sigma^*$  , Even length of strings, strings containing abba are regular Languages.

# Basic Machine and Finite State Machine

- **Basic Machine:**

- A machine that recognizes input set  $I$  and produces output set  $O$ , where  $I$  &  $O$  are finite.
- Machine Function(MAF):  $I \rightarrow O$
- Deals with **what** output is generated, not **how** output is generated

E.g.: Switch

# Basic machine

**It is the most primitive machine with a set of inputs  $I$  and set of output  $O$ .**

E.g. AND,OR,NOR... gates , Vending machine, Decimal to binary converter, Electric fan...

Weighing machine : I/p- Coin

O/p- Printed weight ticket

It just maps input set to output set.

Mapping function is called Machine function MAF  $I \rightarrow O$

E.g.for AND gate MAF:

I	O
(0,0)	0
(0,1)	0
(1,0)	0
(1,1)	1

Limitations:

Cannot remember the intermediate states.

# Finite State machine(FSM)

**In FSM the internal state of machine changes when it receives I/P.**

It has a pair of functions:

Machine Fn **MAF** :  $I \times S \rightarrow O$

State Fn **STF** :  $I \times S \rightarrow S$

Where S: finite or internal state of machine

I: set of i/p symbols

O: set of o/p symbols

For a given i/p there will be always only one final state.

e.g. **Binary Adder, Divisibility by 3 tester..**

# Finite Automata

- 5-tuple:
- $M = (Q, \Sigma, \delta, q_0, F)$  where,
  - $Q$ : Finite set of states
  - $\Sigma$ : Finite input alphabet
  - $\delta$ : STF that maps  $Q \times \Sigma$  to  $Q$ , i.e.  $Q \times \Sigma \longrightarrow Q$
  - $q_0$ : Initial state of FA,  $q_0 \in Q$
  - $F$ : Set of final states,  $F \subseteq Q$



# Finite Automata (cont...)

Input

String

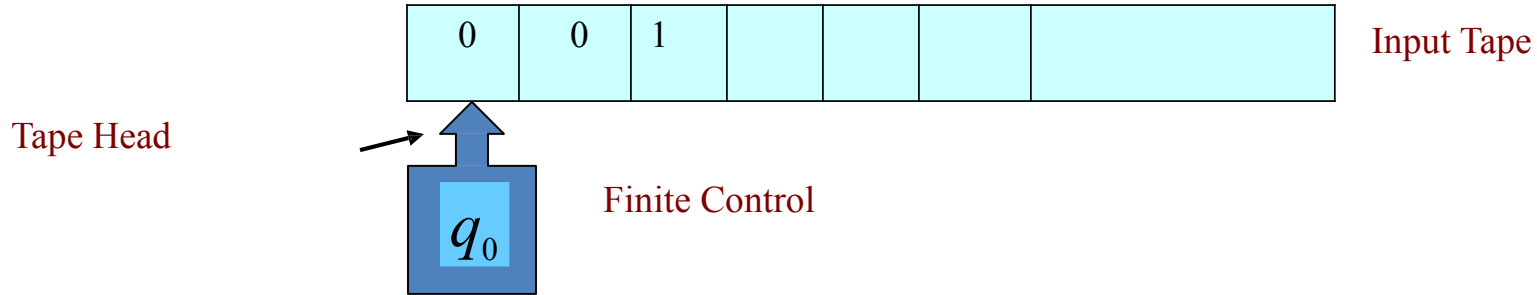


Output



Accept/Reject

# Block diagram of FA



- Tape is divided into units, each containing one symbol of the i/p string from  $\Sigma$ .
- Read the current letter of input under the tape head.
- Transit to a new state depending on the current input and the current state, as dictated by the transition function  $\delta(q_i, a) = q_j$ .
- Halt after consuming the entire input.

# Acceptance of a String

- A string 'x' is said to be accepted by an FA (given by):

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

$$\text{if } \delta(q_0, x) = p,$$

for some  $p \in F$  (i.e.  $p$  is a member of  $F$ )

# Acceptance of a Language

- If there is a language  $L$  such that:

$$L = \{x \mid \delta(q_0, x) = p, \text{ for some } p \in F\},$$

Then it is said to be accepted by the FA,  $M$

# Finite Automata (cont...)

- Finite number of states
- No memory to store intermediate results
- Limited power due to lack of memory
- Input string is considered to be accepted, if the FA resides in any of the final states
- Input string is considered to be rejected, if the FA resides in any of the non-final states



# Preferred Notation for FA

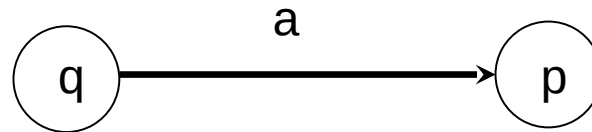
❑ Transition Diagram

❑ Transition Table

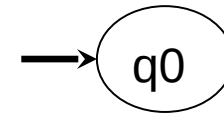
# Transition Diagram

Transition diagram for  $FA(Q, \Sigma, q_0, F, \delta)$  is a graph defined as:

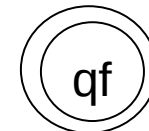
- For each state there is a node.
- For each state  $q$  in  $Q$  and for input symbol in  $\Sigma$ .  
let  $\delta(q, a) = p$  then



- There is arrow into start state.



- Final states marked by double circle.



# Transition Table

- Transition table is a conventional, tabular representation of a function like  $\delta$ (delta) that takes two arguments and return a state.
- The rows of the table correspond to the states, and the columns correspond to the input.
- The entry for one row corresponding to the states  $q$  and the column corresponding to the input  $a$  is the state  $\delta(q,a)$

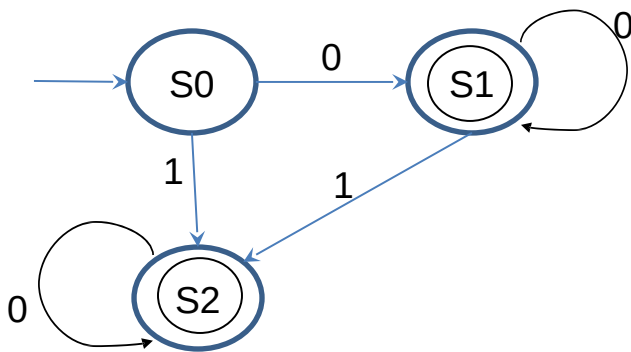
for example:

	0	1
q0	q2	q0
q1	q1	q1
q2	q2	q1



# Deterministic Finite Automata

- An FA is said to be deterministic, if for every state there is a unique input symbol, which takes the state to the required next unique state
- Given a state  $S_j$ , the same input symbol does not cause the FA to move into more than one state-there is always a unique next state for an input symbol

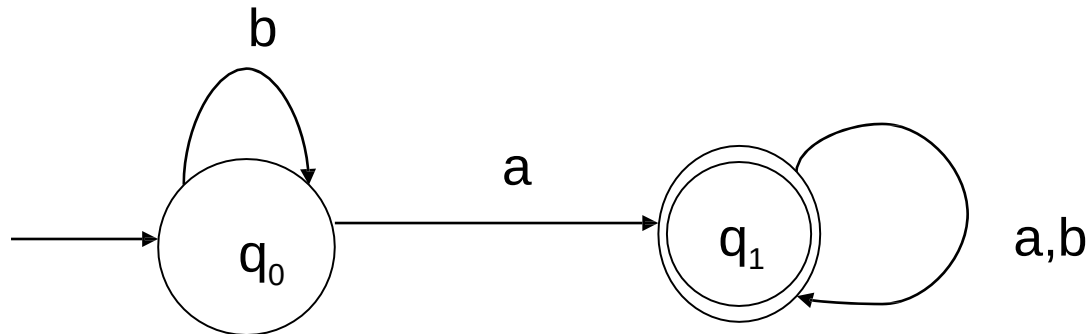


# Designing FA

- Accept strings consisting of a & b, that has at least one 'a'

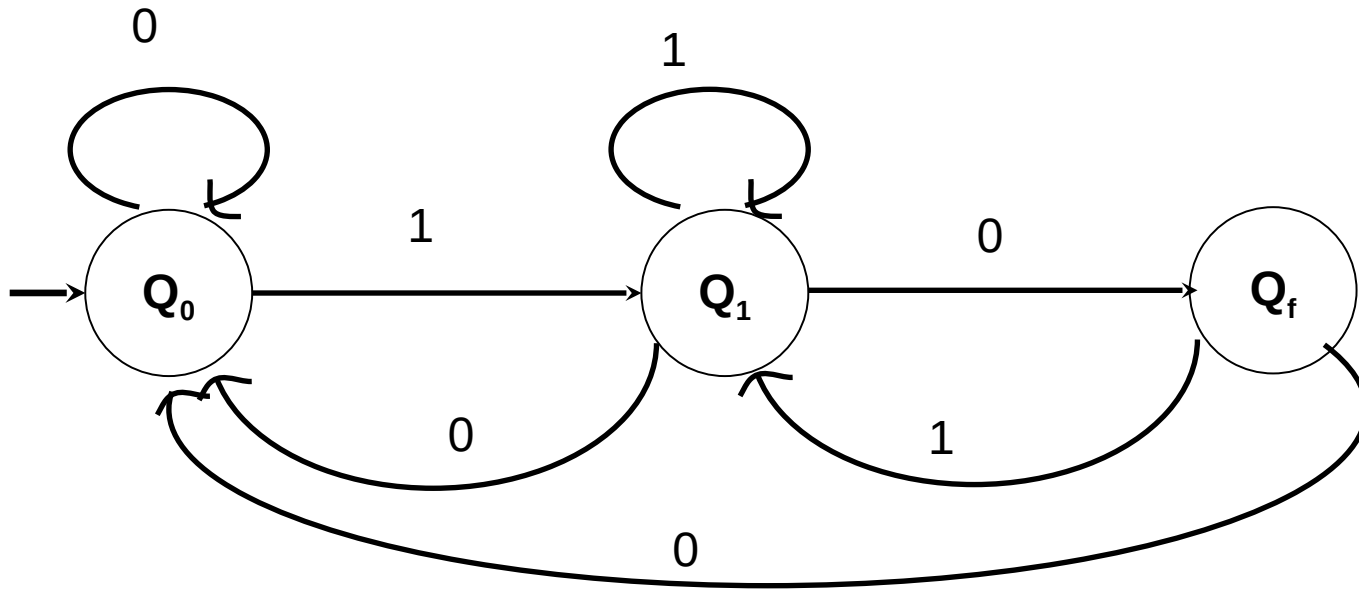
i) Min string :a

ii)  $\Sigma = \{a,b\}$



# Finite Automata Example

□ FA for language over  $\{1,0\}$  in which every string ends with 10.



# FA(DFA & NFA) Examples

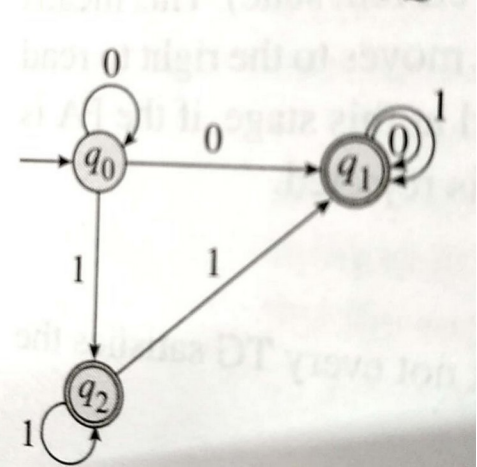
1. Construct a FA to recognize language containing strings starting with '10'
2. Construct a FA to recognize language containing strings ending with '101'
3. Construct a FA to recognize language of strings with exactly two 0s anywhere
4. Construct a FA to recognize language containing strings starting with 'a'
5. Construct a FA to identify string containing even number of 'a's over  $\Sigma = \{a\}$

# Non-Deterministic Finite Automata

- NFA has the power to be in several states at once
- Each NFA accepts a language that is also accepted by some DFA
- NFAs are often easier than DFAs
- We can always convert an NFA to a DFA, but the latter may have exponentially more states than the NFA (a rare case)
- The difference between the DFA and the NFA is the type of transition function  $\delta$
- For a NFA  $\delta$  is a function that takes a state and input symbol as arguments (like the DFA transition function), but returns a set of zero or more states (rather than returning exactly one state, as the DFA must)

# NFA Definition

- 5-tuple:
- $M = (Q, \Sigma, \delta, q_0, F)$  where,
  - $Q$ : Finite set of states
  - $\Sigma$ : Finite input alphabet
  - **$\delta$ : STF that maps  $Q \times \Sigma$  to  $Q$ , i.e.  $Q \times \Sigma \rightarrow 2^Q$**
  - $q_0$ : Initial state of FA,  $q_0 \in Q$
  - $F$ : Set of final states,  $F \subseteq Q$



## Example : pair of 0's or pair of 1's

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

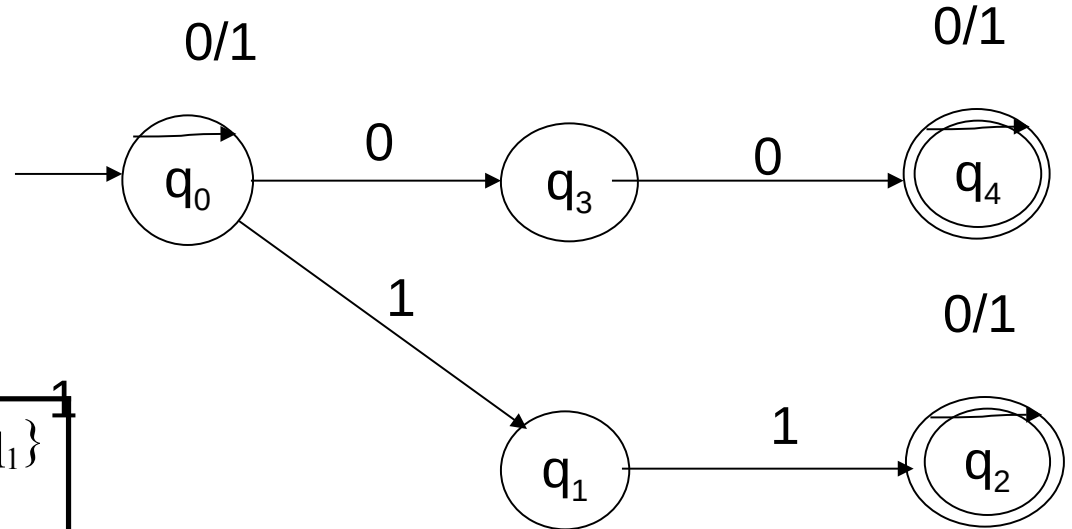
$\Sigma = \{0, 1\}$

Start state is  $q_0$

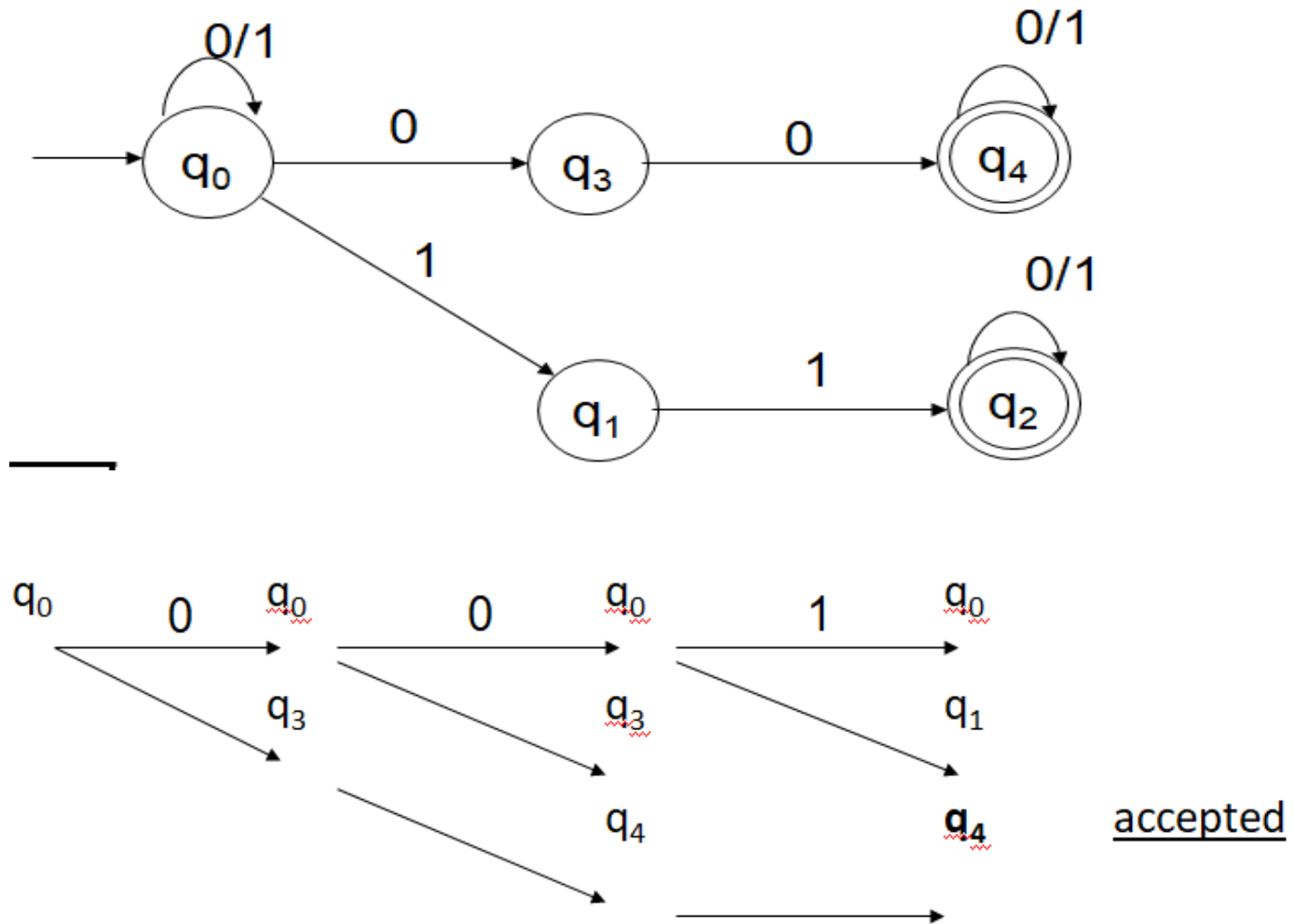
$F = \{q_2, q_4\}$

$\delta$ :

	0	1
$q_0$	$\{q_0, q_3\}$	$\{q_0, q_1\}$
$q_1$	$\{\}$	$\{q_2\}$
$q_2$	$\{q_2\}$	$\{q_2\}$
$q_3$	$\{q_4\}$	$\{\}$
$q_4$	$\{q_4\}$	$\{q_4\}$



# NFA



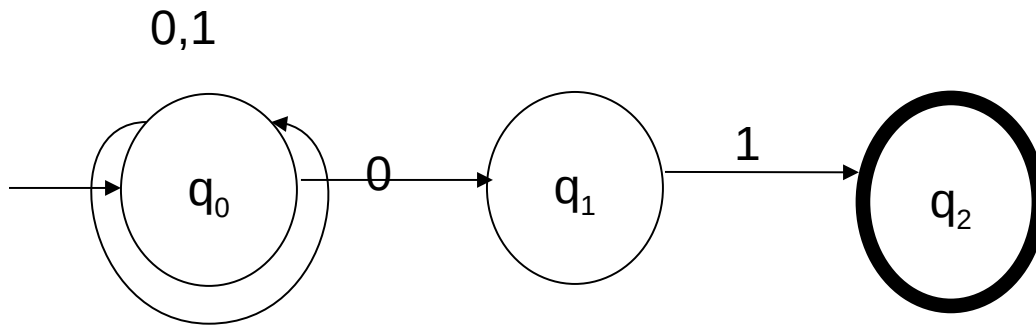


# NFA Examples

1. Construct a FA to recognize language containing strings starting with '10'
2. Construct a FA to recognize language containing strings ending with '101'
3. Construct a FA to recognize language of strings with exactly two 0s anywhere
4. Construct a FA that recognizes the language of strings that end in 01

# NFA

Example: NFA that recognizes the language of strings that end in 01



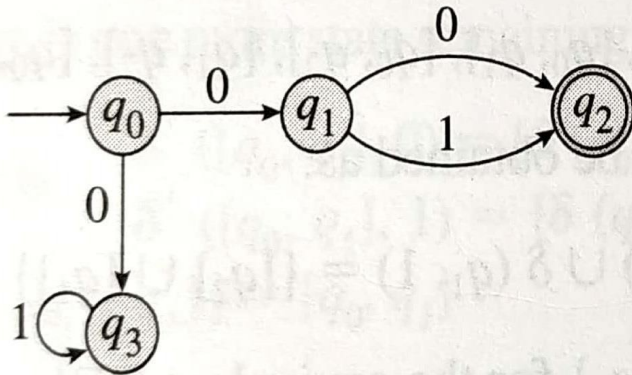
# Equivalence of NFA and DFA

For every NFA, there exists an equivalent DFA.

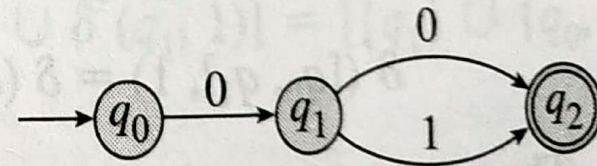
NFA and DFA have equal powers

- Example:**

NFA vs DFA



Example NFA



Equivalent DFA

Identify the language

# Conversion of NFA to DFA

## (Method I)

- For every NFA, there is an equivalent DFA
- **Let us consider  $Q' = 2^Q$ , set of states for resulting DFA**  
 $Q' \times \Sigma \rightarrow Q'$
- **Find all transitions from every state in  $Q'$  on reading each input symbols**
- Every combination of states can be considered as a new state in the resulting DFA and can be given a new label

# Conversion of NFA to DFA

## Method I (cont...)

- In the resulting DFA, there should be a unique state resulting from an input to the previous state
- The initial state remains unchanged for the resultant DFA
- **All possible subsets of  $Q$  (i.e.  $2^Q$ ) are the possible states for the resultant DFA**
- Final state in resulting DFA is all the states containing final state of given NFA

# Conversion of NFA to DFA

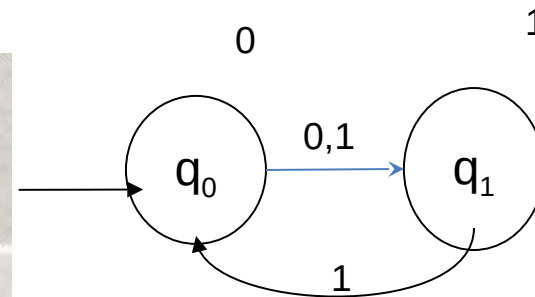
## Method I (cont...)

### Example:

Convert the NFA :  $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$  to its equivalent DFA

State Transition Table  $\delta$  for example NFA

$Q \backslash \Sigma$	0	1
$q_0$	$\{q_0, q_1\}$	$\{q_1\}$
$q_1$	$\phi$	$\{q_0, q_1\}$



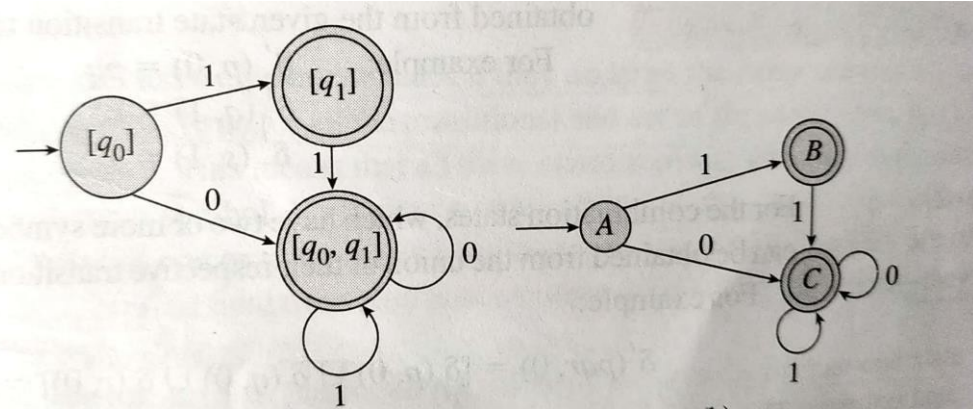
# Conversion of NFA to DFA

## Method I (cont...)

State Transition table for resultant NFA

$Q' \backslash \Sigma$	0	1
$[q_0]$	$[q_0, q_1]$	$[q_1]$
$[q_1]$	—	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$

State Transition Graph for resultant NFA



# Conversion of NFA to DFA

## Method II

- Easier and direct
- Takes lesser number of steps and time to build an equivalent DFA
- Instead of considering the set  $Q' = 2^Q$  and then removing non-required states, consider only those states that are required



# Conversion of NFA to DFA

## Method II (cont...)

- Start building an equivalent DFA with the transition diagram of given NFA
- Find the transition, one state at a time
- If the next state of a given transition is a new combination, add that to the set of states for the resultant DFA

# Conversion of NFA to DFA

## Method II (cont...)

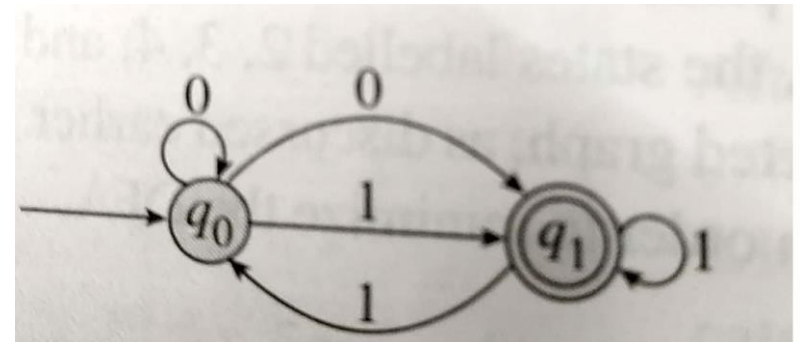
- Example:**

- Convert the NFA:  $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, q_1)$

State Transition Table for example NFA

$Q \backslash \Sigma$	0	1
$q_0$	$\{q_0, q_1\}$	$\{q_1\}$
$q_1$	$\phi$	$\{q_0, q_1\}$

Transition Graph for example NFA

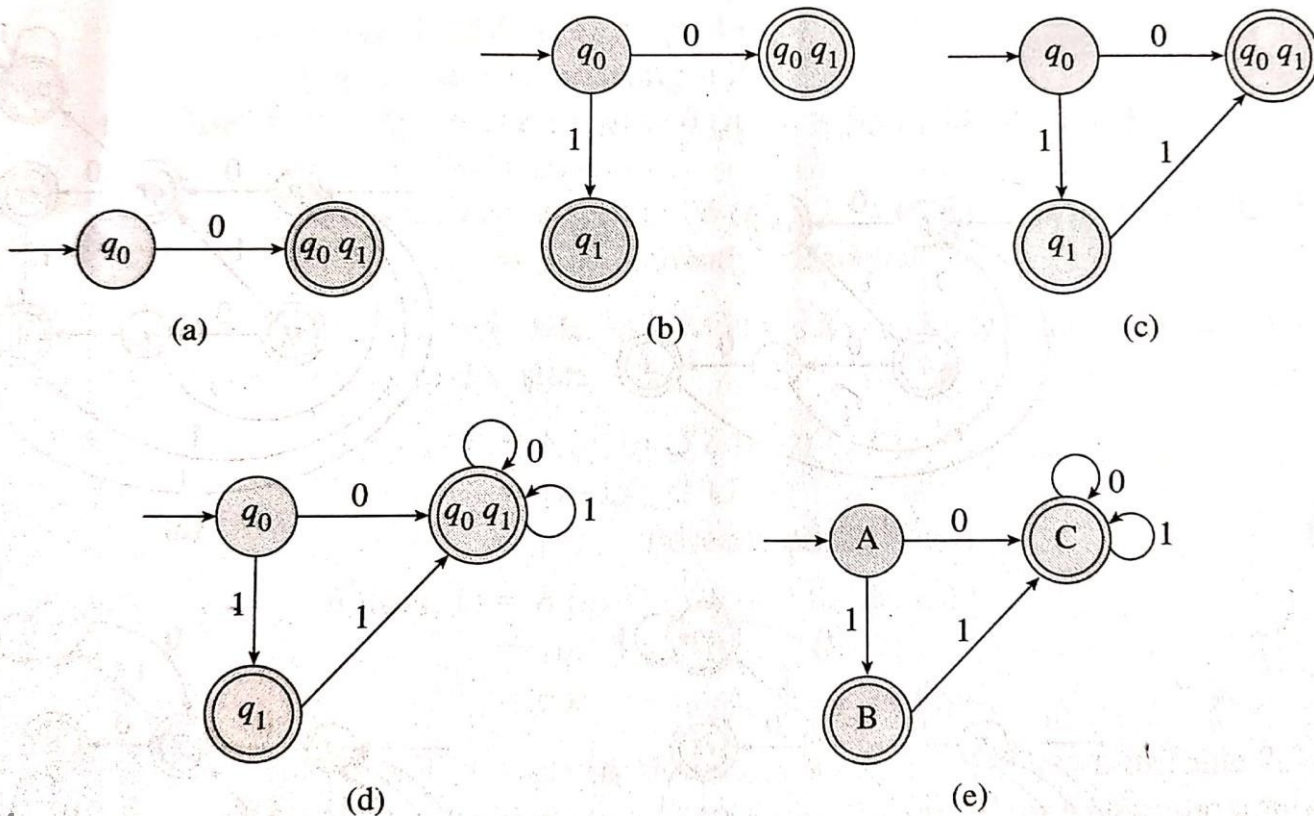


# Conversion of NFA to DFA

## Method II (cont...)

**DFA construction from the given NFA:**

**(a) Step 1 (b) Step 2 (c) Step 3 (d) Step 4 (e) Final DFA**



# DFA Minimization

- Identify equivalent states and keep any one of them.
- Replace remaining states by that one.
- Identify unreachable states and remove them
- Identify dead (or trap) states and remove them

# DFA Minimization (cont..)

- **Equivalent states:**
  - go to same next state on reading the same input symbol
  - same type (final or non-final)
- **Unreachable states:**
  - Can not be reached from initial state on reading any input symbol
- **Dead (or Trap) states:**
  - Have no outgoing transitions, except to themselves

# DFA Minimization Rules

- One non-final state can be replaced by its equivalent non-final state only
- One final state can be replaced by its equivalent final state only
- Initial state can not be replaced by any other state
- One final state can not be replaced by a non-final state or one non-final state can not be replaced by a final state

# DFA Minimization Rules (cont...)

- Replacing state A by state B means deleting all entries related to state A i.e. deleting all the transitions for state A from the state transition table.
- If more equivalent states are found after applying all the rules, repeat the same five steps to further reduce the DFA.

Example

	Q	0	1
Q	<del>Q</del>		
p	<del>Q</del>	p,q	p
q		r	r
r		s	-
s		s	s



# DFA Minimization (cont...)

State Transition Table of the DFA to be minimized

$\Sigma$		0	1
$Q'$			
(1)	$p$	$pq$ (5)	$p$ (1)
(2)	$q$	$r$ (3)	$r$ (3)
(3)	$r$	$s$ (4)	—
*(4)	$s$	$s$ (4)	$s$ (4)
(5)	$pq$	$pqr$ (11)	$pr$ (6)
(6)	$pr$	$pqs$ (12)	$p$ (1)
*(7)	$ps$	$pqs$ (12)	$ps$ (7)
(8)	$qr$	$rs$ (10)	$r$ (3)
*(9)	$qs$	$rs$ (10)	$rs$ (10)
*(10)	$rs$	$s$ (4)	$s$ (4)
(11)	$pqr$	$pqrs$ (15)	$pr$ (6)
*(12)	$pqs$	$pqrs$ (15)	$prs$ (13)
×*(13)	$prs$	$pqs$ (12)	$ps$ (7)
×*(14)	$qrs$	$rs$ (10)	$rs$ (10)
×*(15)	$pqrs$	$pqrs$ (15)	$prs$ (13)

↑  
New labels

\*: Final states



# DFA Minimization (cont...)

## Minimized DFA

$Q' \backslash \Sigma$	0	1
1	5	1
2	3	3
3	4	—
*4	4	4
5	11	6
6	12	1
*7	12	7
8	4 •	3
*9	4 •	4 •
11	12 •	6
*12	12 •	7 •

‘\*’: Final states

‘•’: Modified entries due to replacement

# DFA Minimization (cont...)

## Further minimized DFA

$Q' \backslash \Sigma$	0	1
1	5	1
2	3	3
3	4	—
*4	4	4
5	11	6
6	7 •	1
*7	7 •	7
8	4	3
11	7 •	6

‘\*’: Final states

‘•’: Modified entries due to replacement

# NFA with $\varepsilon$ - Transitions

- 5-tuple:
- $M = (Q, \Sigma, \delta, q_0, F)$  where,
  - $Q$ : Finite set of states
  - $\Sigma$ : Finite input alphabet
  - **$\delta$ : STF that maps  $Q \times (\Sigma \cup \{\varepsilon\})$  to  $2^Q$**

**i.e.  $Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$**

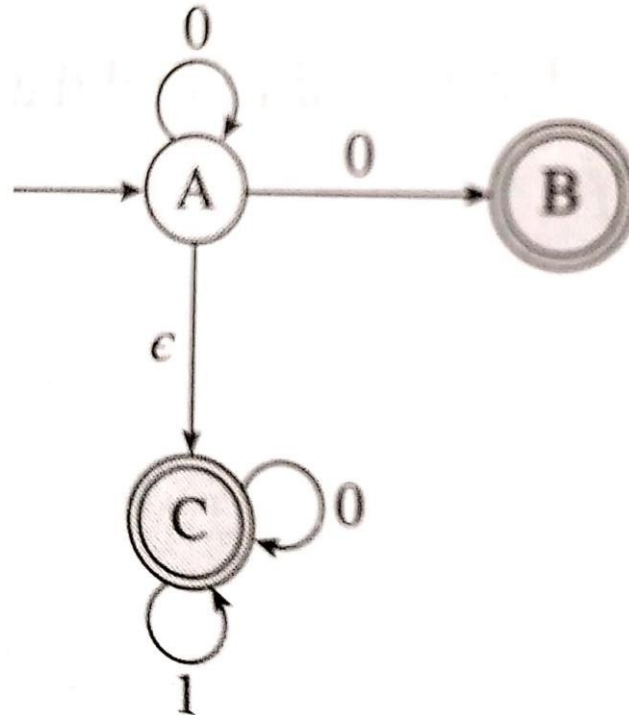
- $q_0$ : Initial state of FA,  $q_0 \in Q$
- $F$ : Set of final states,  $F \subseteq Q$

# NFA with $\epsilon$ - Transitions

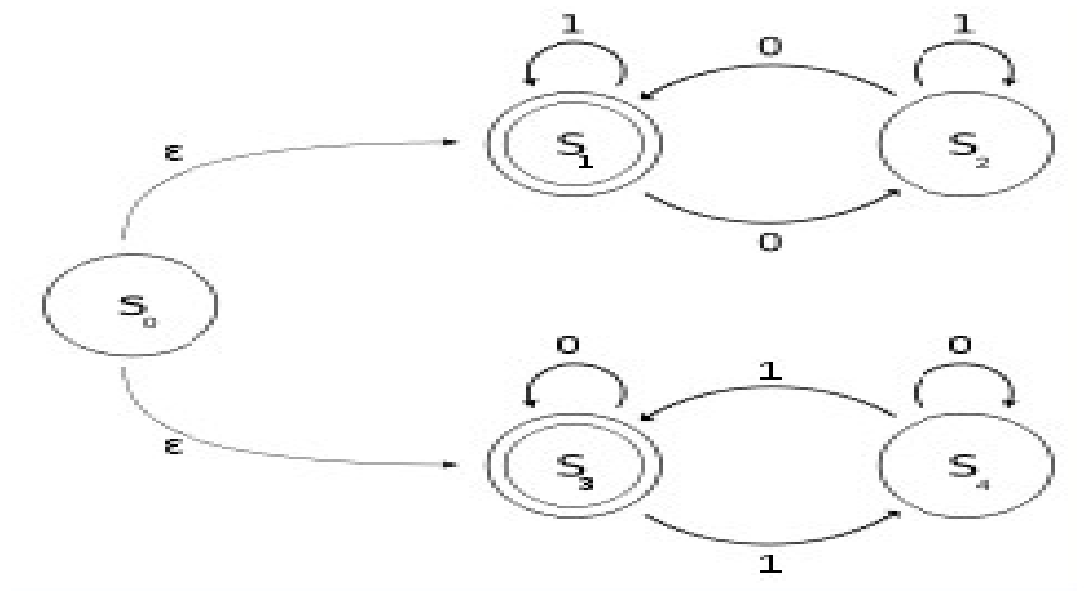
(cont..)

- Example:**

Transition Graph for example NFA with  $\epsilon$  - transitions



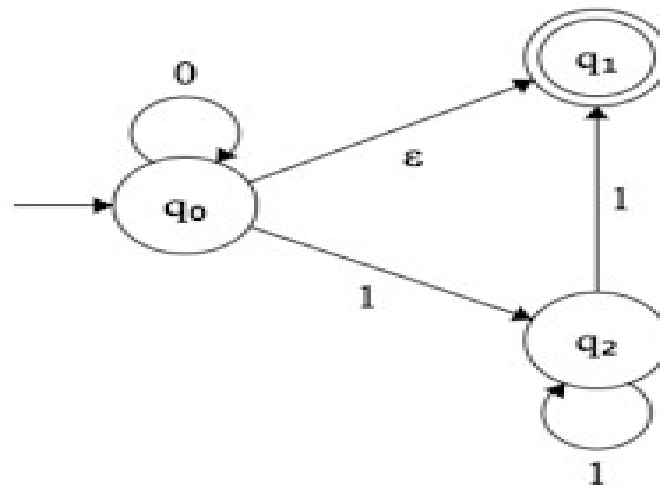
# NFA with $\epsilon$ moves Example



# NFA with $\epsilon$ - Transitions (Examples)

- Example:**

Transition Graph for example NFA with  $\epsilon$  - transitions



# NFA with $\epsilon$ - Transitions (Examples)

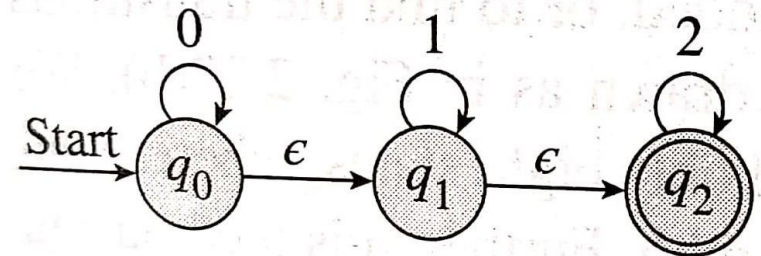
- Example:**

Convert the NFA with  $\epsilon$  -transitions given in the figure to its equivalent DFA

State Transition table for example NFA with  $\epsilon$  -transitions

$Q \backslash \Sigma \cup \{\epsilon\}$	0	1	2	$\epsilon$
$q_0$	$\{q_0\}$	$\phi$	$\phi$	$\{q_1\}$
$q_1$	$\phi$	$\{q_1\}$	$\phi$	$\{q_2\}$
$q_2$	$\phi$	$\phi$	$\{q_2\}$	$\phi$

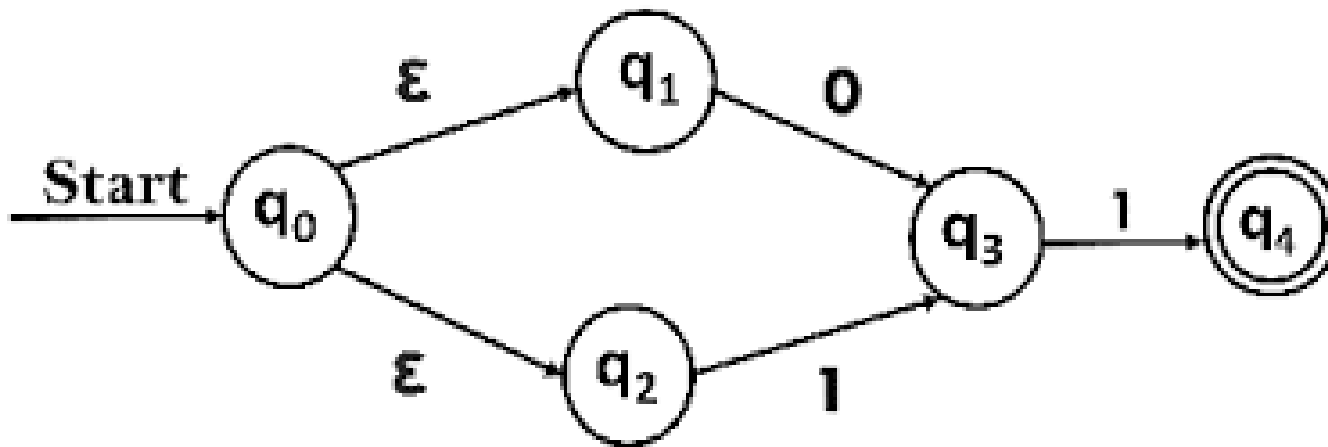
State Transition Graph for example NFA with  $\epsilon$  -transitions



# NFA with $\epsilon$ - Transitions (Examples)

- Example:**

Transition Graph for example NFA with  $\epsilon$  - transitions



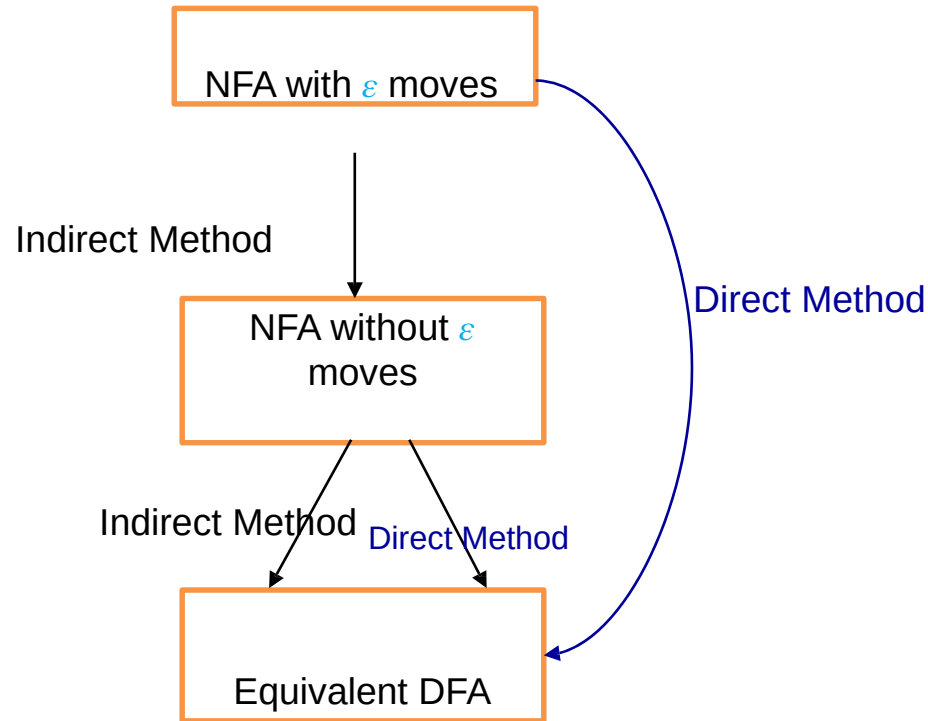


# Significance of NFA with $\epsilon$ - Transitions

- Helps to split complex language acceptance problems into smaller ones.
- Solution to these problems can be integrated with the help of  $\epsilon$  - transition.
- Concatenation ,parallel path can be connected with the  $\epsilon$  - transition.

# Conversion of NFA with $\epsilon$ - Transitions to DFA

- Indirect Conversion Method
- Direct Conversion Method



# Indirect Conversion Method

1. Construct the NFA without  $\varepsilon$  - *transitions* from the given NFA with  $\varepsilon$  - move
2. Construct an equivalent DFA with this newly constructed NFA without  $\varepsilon$  - *transitions* (using existing methods)

# Indirect Conversion Method

## (cont...)

- $\epsilon$ -closure of a state:
  - Set of all states  $p$ , such that there is a path from state  $q$  to state  $p$  labeled ' $\epsilon$ ', is known as  $\epsilon$ -closure ( $q$ )
  - Set of all the states having distance zero from state  $q$
  - Every state is at distance zero from itself
  - Denoted by  $\delta^{\wedge}$

# Indirect Conversion Method

## (cont...)

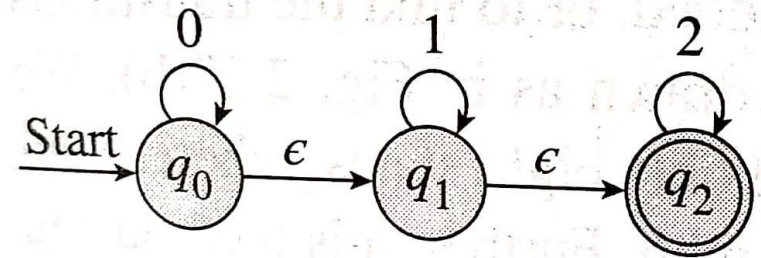
- Example:**

Convert the NFA with  $\epsilon$  -transitions given in the figure to its equivalent DFA

State Transition table for example NFA with  $\epsilon$  -transitions

$Q \backslash \Sigma \cup \{\epsilon\}$	0	1	2	$\epsilon$
$q_0$	$\{q_0\}$	$\phi$	$\phi$	$\{q_1\}$
$q_1$	$\phi$	$\{q_1\}$	$\phi$	$\{q_2\}$
$q_2$	$\phi$	$\phi$	$\{q_2\}$	$\phi$

State Transition Graph for example NFA with  $\epsilon$  -transitions

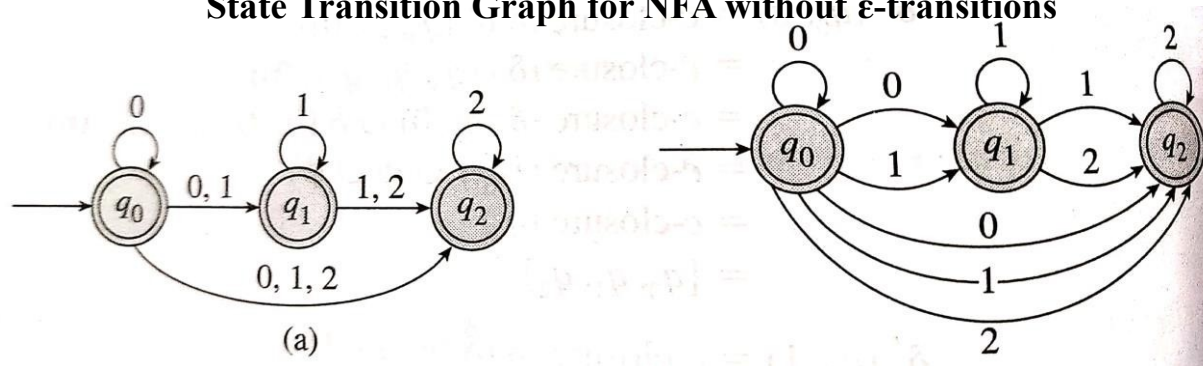


# Indirect Conversion Method

## (cont...)

### 1. Conversion of NFA with $\epsilon$ -transitions to NFA without $\epsilon$ -transitions:

State Transition Graph for NFA without  $\epsilon$ -transitions



State Transition Table for NFA without  $\epsilon$ -transitions

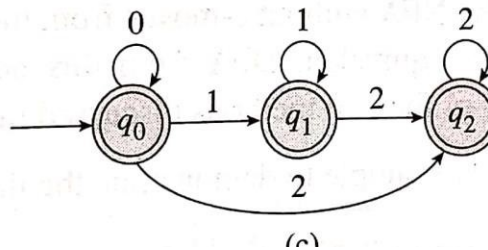
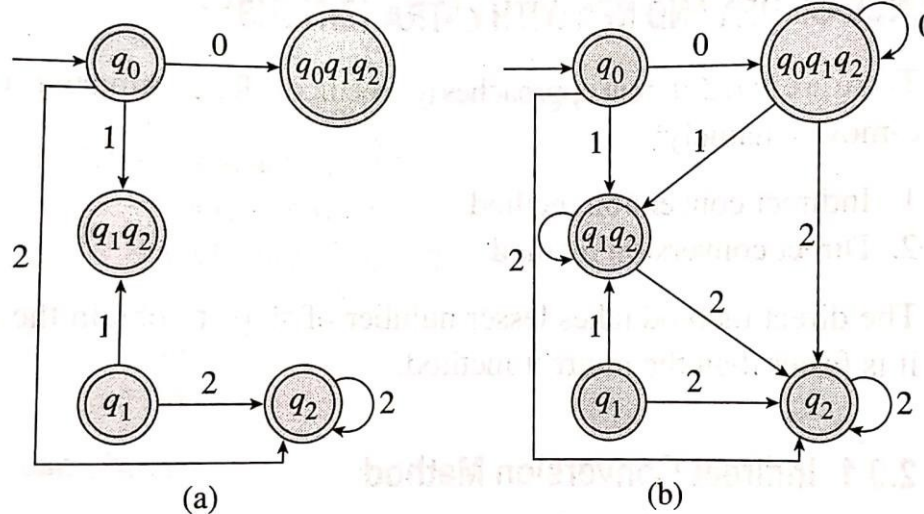
$Q \backslash \Sigma$	0	1	2
$q_0$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
$q_1$	$\phi$	$\{q_1, q_2\}$	$\{q_2\}$
$q_2$	$\phi$	$\phi$	$\{q_2\}$

# Indirect Conversion Method

## (cont...)

### 2. Conversion of NFA without $\epsilon$ -transitions to an equivalent DFA:

(a) Step 1 (b) Step 2 (c) Final DFA





# Indirect Conversion Method

## (cont...)

## 2. Conversion of NFA without $\epsilon$ -transitions to an equivalent DFA:

State Transition table for the equivalent DFA

$Q' \backslash \Sigma$	0	1	2
$*q_0$	$q_0q_1q_2$	$q_1q_2$	$q_2$
$*q_1$	—	$q_1q_2$	$q_2$
$*q_2$	—	—	$q_2$
$*q_1q_2$	—	$q_1q_2$	$q_2$
$*q_0q_1q_2$	$q_0q_1q_2$	$q_1q_2$	$q_2$

\*: Final states

Reduced State Transition table for the equivalent DFA

$Q' \backslash \Sigma$	0	1	2
$*q_0$	$q_0 \bullet$	$q_1 \bullet$	$q_2$
$*q_1$	—	$q_1 \bullet$	$q_2$
$*q_2$	—	—	$q_2$

\*: Final states

•: Modified entries



# Direct Conversion Method

- Begin with the initial state
- Go on adding the states as & when required to the diagram
- Follow the same process until there exists no state without having the transitions specified
- Each state label consists of:
  - Name of state label
  - Combination of all the state symbols, which are reachable from the given state

# Direct Conversion Method

(cont...)

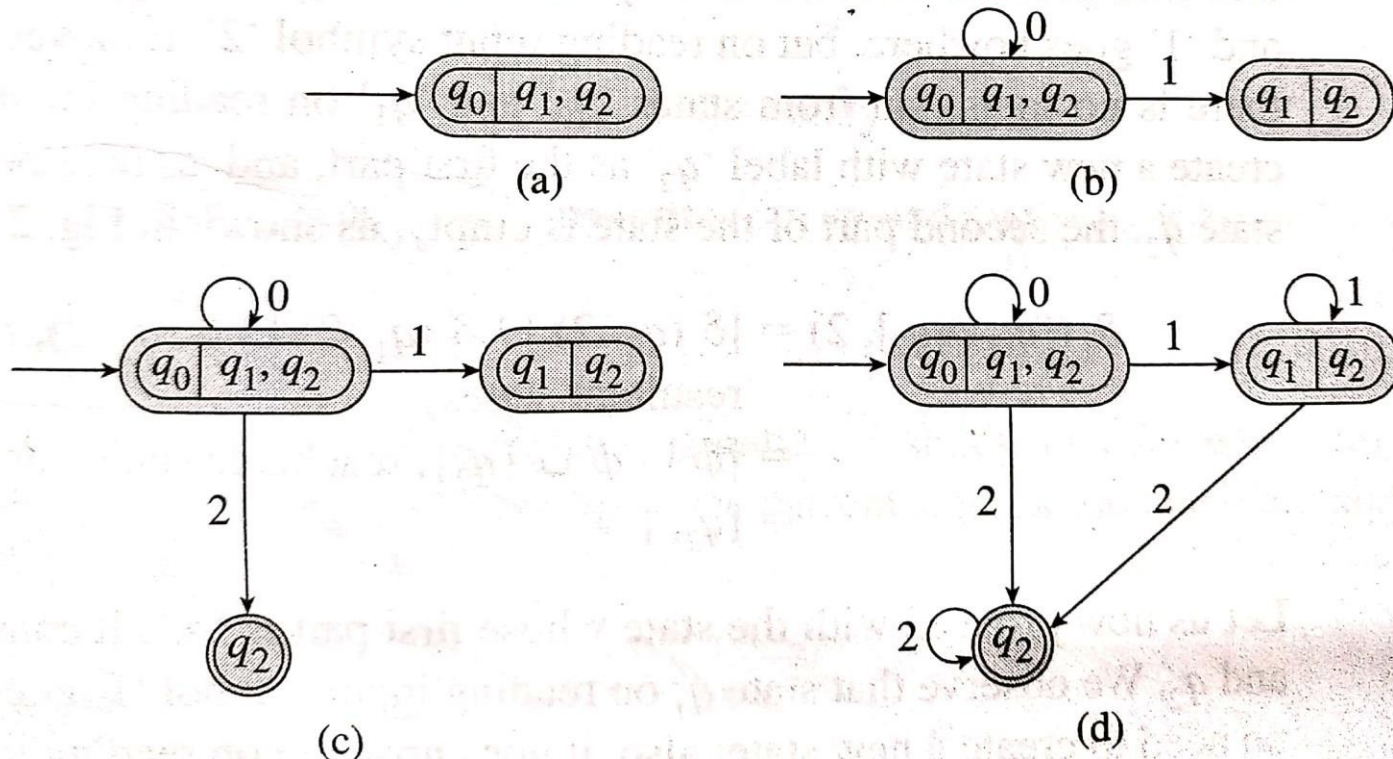
- Start building an equivalent DFA with the transition diagram of given NFA
- Find the transition, one state at a time
- If the next state of a given transition is a new combination, add that to the set of states for the resultant DFA

# Direct Conversion Method

## (cont...)

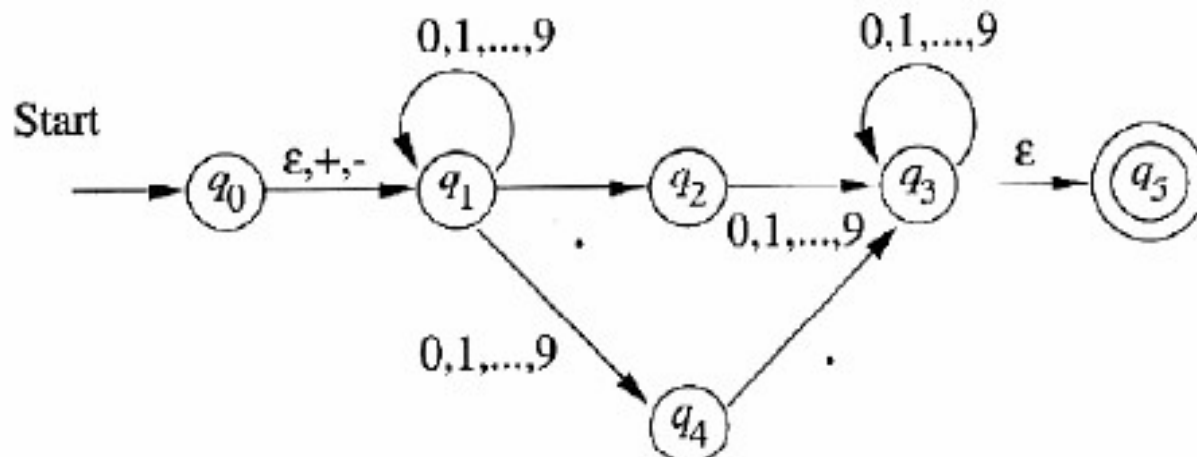
- Example (Refer slide no. 53 for example NFA with  $\varepsilon$ -transitions)

Resultant DFA (a) Step 1 (b) Step 2 (c) Step 3 (d) Final DFA



**Example 2.16:** In Fig. 2.18 is an  $\epsilon$ -NFA that accepts decimal numbers consisting of:

1. An optional  $+$  or  $-$  sign,
2. A string of digits,
3. A decimal point, and
4. Another string of digits. Either this string of digits, or the string (2) can be empty, but at least one of the two strings of digits must be nonempty.



**Example 2.18:** The  $\epsilon$ -NFA of Fig. 2.18 is represented formally as

$$E = (\{q_0, q_1, \dots, q_5\}, \{., +, -, 0, 1, \dots, 9\}, \delta, q_0, \{q_5\})$$

where  $\delta$  is defined by the transition table in Fig. 2.20.  $\square$

	$\epsilon$	$+, -$	$.$	$0, 1, \dots, 9$
$q_0$	$\{q_1\}$	$\{q_1\}$	$\emptyset$	$\emptyset$
$q_1$	$\emptyset$	$\emptyset$	$\{q_2\}$	$\{q_1, q_4\}$
$q_2$	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_3\}$
$q_3$	$\{q_5\}$	$\emptyset$	$\emptyset$	$\{q_3\}$
$q_4$	$\emptyset$	$\emptyset$	$\{q_3\}$	$\emptyset$
$q_5$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Figure 2.20: Transition table for Fig. 2.18

**Example 2.20:** Let us compute  $\hat{\delta}(q_0, 5.6)$  for the  $\epsilon$ -NFA of Fig. 2.18. A summary of the steps needed are as follows:

- $\hat{\delta}(q_0, \epsilon) = \text{ECLOSE}(q_0) = \{q_0, q_1\}$ .
- Compute  $\hat{\delta}(q_0, 5)$  as follows:
  1. First compute the transitions on input 5 from the states  $q_0$  and  $q_1$  that we obtained in the calculation of  $\hat{\delta}(q_0, \epsilon)$ , above. That is, we compute  $\delta(q_0, 5) \cup \delta(q_1, 5) = \{q_1, q_4\}$ .
  2. Next,  $\epsilon$ -close the members of the set computed in step (1). We get  $\text{ECLOSE}(q_1) \cup \text{ECLOSE}(q_4) = \{q_1\} \cup \{q_4\} = \{q_1, q_4\}$ . That set is  $\hat{\delta}(q_0, 5)$ . This two-step pattern repeats for the next two symbols.
- Compute  $\hat{\delta}(q_0, 5.)$  as follows:
  1. First compute  $\delta(q_1, .) \cup \delta(q_4, .) = \{q_2\} \cup \{q_3\} = \{q_2, q_3\}$ .
  2. Then compute
 
$$\hat{\delta}(q_0, 5.) = \text{ECLOSE}(q_2) \cup \text{ECLOSE}(q_3) = \{q_2\} \cup \{q_3, q_5\} = \{q_2, q_3, q_5\}$$
- Compute  $\hat{\delta}(q_0, 5.6)$  as follows:
  1. First compute  $\delta(q_2, 6) \cup \delta(q_3, 6) \cup \delta(q_5, 6) = \{q_3\} \cup \{q_3\} \cup \emptyset = \{q_3\}$ .
  2. Then compute  $\hat{\delta}(q_0, 5.6) = \text{ECLOSE}(q_3) = \{q_3, q_5\}$ .

### 2.5.5 Eliminating $\epsilon$ -Transitions

Given any  $\epsilon$ -NFA  $E$ , we can find a DFA  $D$  that accepts the same language as  $E$ . The construction we use is very close to the subset construction, as the states of  $D$  are subsets of the states of  $E$ . The only difference is that we must incorporate  $\epsilon$ -transitions of  $E$ , which we do through the mechanism of the  $\epsilon$ -closure.

Let  $E = (Q_E, \Sigma, \delta_E, q_0, F_E)$ . Then the equivalent DFA

$$D = (Q_D, \Sigma, \delta_D, q_D, F_D)$$

is defined as follows:

1.  $Q_D$  is the set of subsets of  $Q_E$ . More precisely, we shall find that all accessible states of  $D$  are  $\epsilon$ -closed subsets of  $Q_E$ , that is, sets  $S \subseteq Q_E$  such that  $S = \text{ECLOSE}(S)$ . Put another way, the  $\epsilon$ -closed sets of states  $S$  are those such that any  $\epsilon$ -transition out of one of the states in  $S$  leads to a state that is also in  $S$ . Note that  $\emptyset$  is an  $\epsilon$ -closed set.
2.  $q_D = \text{ECLOSE}(q_0)$ ; that is, we get the start state of  $D$  by closing the set consisting of only the start state of  $E$ . Note that this rule differs from the original subset construction, where the start state of the constructed automaton was just the set containing the start state of the given NFA.
3.  $F_D$  is those sets of states that contain at least one accepting state of  $E$ . That is,  $F_D = \{S \mid S \text{ is in } Q_D \text{ and } S \cap F_E \neq \emptyset\}$ .
4.  $\delta_D(S, a)$  is computed, for all  $a$  in  $\Sigma$  and sets  $S$  in  $Q_D$  by:
  - (a) Let  $S = \{p_1, p_2, \dots, p_k\}$ .
  - (b) Compute  $\bigcup_{i=1}^k \delta_E(p_i, a)$ ; let this set be  $\{r_1, r_2, \dots, r_m\}$ .
  - (c) Then  $\delta_D(S, a) = \bigcup_{j=1}^m \text{ECLOSE}(r_j)$ .

Since the start state of  $E$  is  $q_0$ , the start state of  $D$  is  $\text{ECLOSE}(q_0)$ , which is  $\{q_0, q_1\}$ . Our first job is to find the successors of  $q_0$  and  $q_1$  on the various symbols in  $\Sigma$ ; note that these symbols are the plus and minus signs, the dot, and the digits 0 through 9. On  $+$  and  $-$ ,  $q_1$  goes nowhere in Fig. 2.18, while  $q_0$  goes to  $q_1$ . Thus, to compute  $\delta_D(\{q_0, q_1\}, +)$  we start with  $\{q_1\}$  and  $\epsilon$ -close it. Since there are no  $\epsilon$ -transitions out of  $q_1$ , we have  $\delta_D(\{q_0, q_1\}, +) = \{q_1\}$ . Similarly,  $\delta_D(\{q_0, q_1\}, -) = \{q_1\}$ . These two transitions are shown by one arc in Fig. 2.22.

Next, we need to compute  $\delta_D(\{q_0, q_1\}, \cdot)$ . Since  $q_0$  goes nowhere on the dot, and  $q_1$  goes to  $q_2$  in Fig. 2.18, we must  $\epsilon$ -close  $\{q_2\}$ . As there are no  $\epsilon$ -transitions out of  $q_2$ , this state is its own closure, so  $\delta_D(\{q_0, q_1\}, \cdot) = \{q_2\}$ .

Finally, we must compute  $\delta_D(\{q_0, q_1\}, 0)$ , as an example of the transitions from  $\{q_0, q_1\}$  on all the digits. We find that  $q_0$  goes nowhere on the digits, but  $q_1$  goes to both  $q_1$  and  $q_4$ . Since neither of those states have  $\epsilon$ -transitions out, we conclude  $\delta_D(\{q_0, q_1\}, 0) = \{q_1, q_4\}$ , and likewise for the other digits.



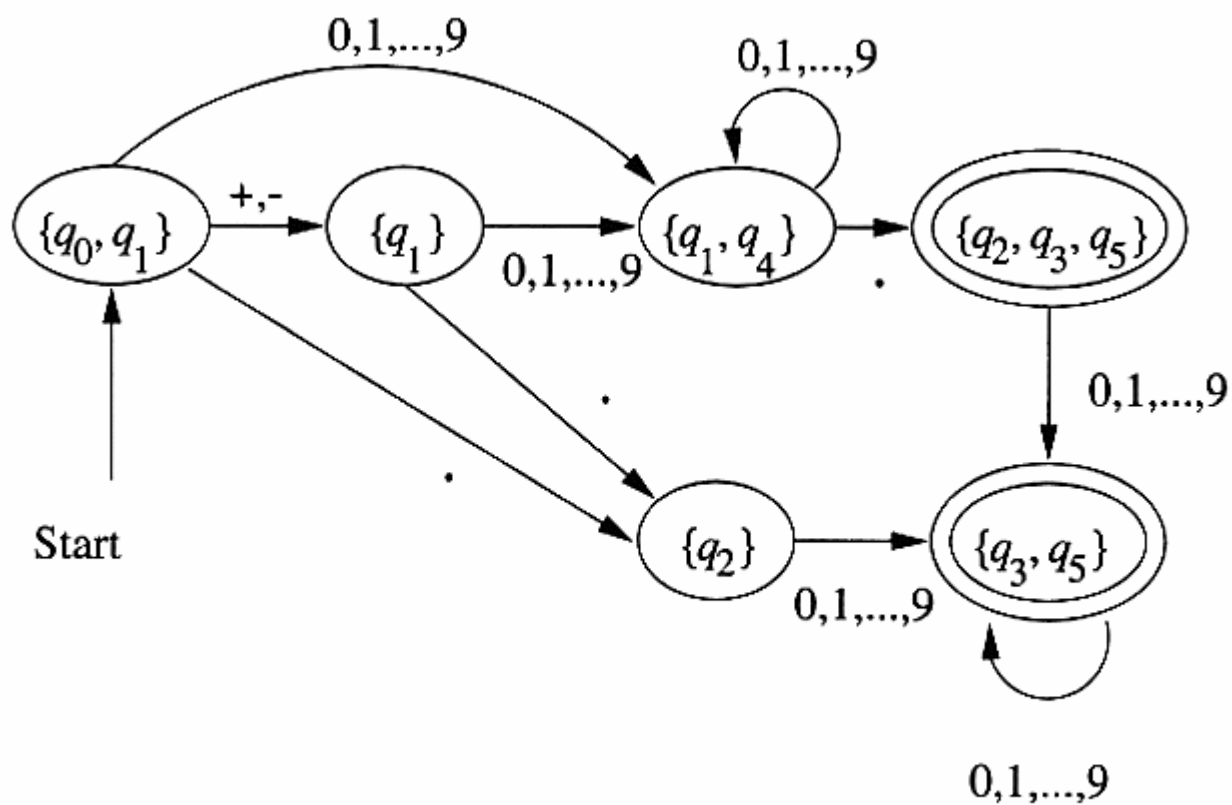


Figure 2.22: The DFA  $D$  that eliminates  $\epsilon$ -transitions from Fig. 2.18

# Convert $\epsilon$ -NFA to DFA example

	a	b	c	$\epsilon$
p	{P}	{q}	{r}	$\emptyset$
q	{q}	{r}	$\emptyset$	{p}
*r	{r}	$\emptyset$	{p}	{q}

Solution  
NFA

	a	b	c	$\epsilon$ -closure(p) = { p } $\epsilon$ -closure(p) = { p, q } $\epsilon$ -closure(p) = { p, q, r }			
p	{P}	{p, q}	{p, q, r}	DFA using direct method			
q	{p, q}	{p, q, r}	{p, q, r}				
*r	{p, q, r}	{p, q, r}	{p, q, r}				
				[p]	[p]	[pq]	[pqr]
				[pq]	[pq]	[pqr]	[pqr]
				*[pqr]	[pqr]	[pqr]	[pqr]

# Convert $\epsilon$ -NFA to DFA example

	0	1	$\epsilon$
A	{A}	$\emptyset$	{B}
B	{C}	$\emptyset$	{D}
C	$\emptyset$	{B}	{D}
*D	{D}	{D}	$\emptyset$

Solution

NFA

	0	1	DFA using direct method		
A	{A,B,C,D}	{D}		0	1
B	{C,D}	{D}	[A]	[ABCD]	[D]
C	---	{B,D}	[ABCD]	[ABCD]	[BD]
*D	{D}	{D}	*[D]	-[D]	[D]
			[BD]	[CD]	[D]
			[CD]	[D]	[BD]

# Applications and Limitations of FA

- **Finite Automata:**
  - Software for designing and checking behavior of digital circuits
  - The “Lexical Analyzer” of a compiler
  - Software for scanning large bodies of text
    - E.g. to find occurrences of words in collections of web pages*
  - Software for verifying systems of all types that have a finite number of distinct states
    - E.g. communication Protocols*

# Applications and Limitations of FA

- Use in Text Editor and Spell Checkers.
- **Limitations:**
  - Cannot recognize Palindrome of string.
  - Set of strings over "(" and ")" & have balanced parenthesis.
  - No Memory, Only memory it has is State and State.

# Application of TOC

- Design of the “Lexical Analyzer” of a compiler.
- Use of the Mealy and Moore Machines for designing the combination and sequential circuits.
- Used in diverse fields, including the management of tap water, Pharmaceutical field, Environmental Field etc..
- CFG n PDA can be used in defining n implementing programming languages.
- So on many more...

# Application of TOC (cont...)

- **Turing Machine:**
  - Prototype/Abstract of modern computer
  - To find out what a computer can do and what it can not do
  - Help us understand what we can expect from our software
  - To decide, whether we can solve the problem, which requires more time to solve, in less amount of time.

Thank You...!!!