

# Regular languages and finite automata

## Study Guide

**Prof. Riddhi Atulkumar Mehta**  
CSE, PIT  
Parul University

1. Regular expressions and languages.....	1
2. Deterministic Finite Automata -(DFA) and equivalence with regular expressions.....	2
3. Moore machines and mealy machines, Conversion from Mealy to Moore and vice versa ....	2
4. Non-Deterministic Finite automata (NFA) and equivalence with DFA.....	3
5. regular grammars and equivalence with finite automata.....	4
6. properties of regular languages, pumping lemma for regular languages.....	4
7. minimization of finite automata.....	5

## 2.1 Regular Expressions and Languages Grammars

### Regular Expressions

Regular expressions (REs) are symbolic notations used to represent *regular languages*. They are built using a finite alphabet and operators.

#### Purpose:

To specify regular languages, which are the simplest class in the Chomsky hierarchy.

#### Syntax:

##### Given an alphabet $\Sigma$ :

- $\emptyset$  denotes the empty language
- $\epsilon$  denotes the empty string
- $a \in \Sigma$  denotes the language  $\{a\}$
- If  $r$  and  $s$  are regular expressions, then:
  - $r + s$  denotes the union (alternation)
  - $rs$  denotes concatenation
  - $r^*$  denotes the Kleene star (zero or more repetitions)

#### Example:

For  $\Sigma = \{0, 1\}$ , the RE  $0^*1^*$  represents all strings with all 0s followed by all 1s (e.g.,  $\epsilon$ , 0, 1, 00011).

### Regular Languages

A language is regular if it can be expressed using a regular expression or accepted by a finite automaton.

#### Example:

$R = (ab)^*$

$\rightarrow L(R) = \{\epsilon, ab, abab, ababab, \dots\}$

#### Closure Properties:

Regular languages are closed under:

Operation	Explanation
Union	$L_1 \cup L_2$ is regular
Concatenation	$L_1L_2$ is regular

Kleene Star	$L^*$ is regular
Complementation	If $L$ is regular, so is its complement
Intersection	$L_1 \cap L_2$ is regular
Difference	$L_1 - L_2$ is regular

### Applications of Regular Expressions

- Text editors (search and replace)
- Lexical analyzers (tokenization in compilers)
- Pattern matching in programming (e.g., Python, JavaScript, grep)
- Validation (email, password formats)

## 2.2 Deterministic Finite Automata (DFA) and Equivalence with Regular Expressions

### Deterministic Finite Automaton (DFA)

A DFA is a mathematical model for a machine that accepts regular languages with deterministic transitions.

#### Formal Definition:

A DFA is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$  where:

- $Q$ : finite set of states
- $\Sigma$ : input alphabet
- $\delta$ : transition function,  $\delta: Q \times \Sigma \rightarrow Q$
- $q_0 \in Q$ : start state
- $F \subseteq Q$ : set of accepting (final) states

### DFA Example

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

Language: Strings ending with '01'

States:  $Q = \{q_0, q_1, q_2\}$

Start state:  $q_0$

Accept state:  $q_2$

Transitions:

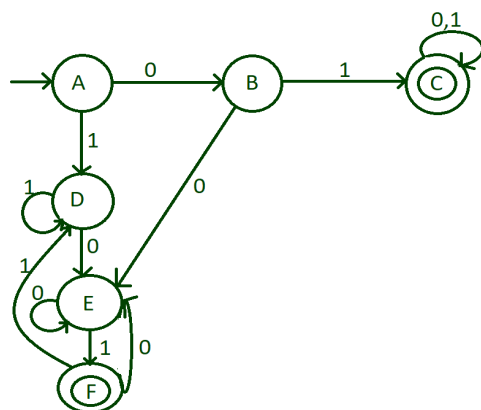
$$\delta(q_0, 0) = q_0$$

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

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

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

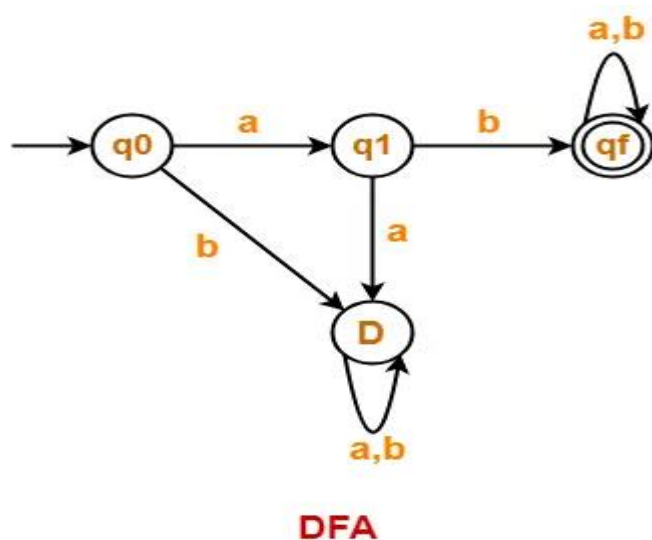
$$\delta(q_2, 0/1) = q_0 \text{ (or dead state)}$$



#### DFA Example

Draw a DFA for the language accepting strings starting with 'ab' over input alphabets  $\Sigma = \{a, b\}$

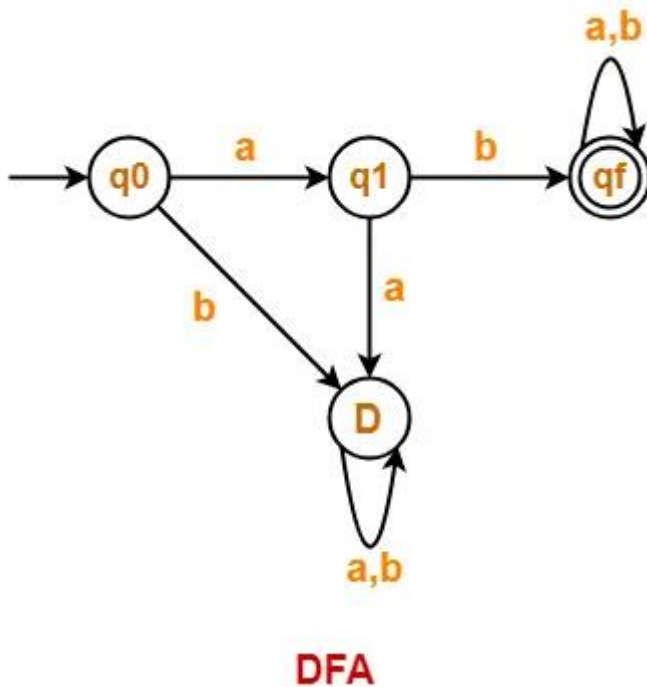
Regular expression for the given language =  $ab(a + b)^*$



#### DFA Example

Draw a DFA for the language accepting strings starting with 'a' over input alphabets  $\Sigma = \{a, b\}$

Regular expression for the given language =  $a(a + b)^*$



### Equivalence to Regular Expressions

- **Kleene's Theorem:**
  - Every regular expression has an equivalent DFA.
  - Every DFA can be converted into an equivalent regular expression.

### Conversion Techniques:

1. RE to NFA (Thompson's Construction)  $\rightarrow$  NFA to DFA
2. DFA to RE (State Elimination Method)

## 2.3 Moore and Mealy Machines

### Moore Machine

- A finite state machine where output is associated with *states*.

### Definition:

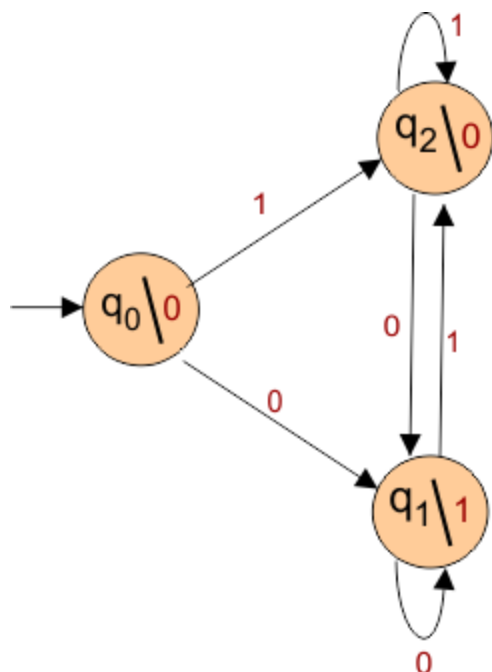
A 6-tuple  $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ , where:

- $Q$ : set of states
- $\Sigma$ : input alphabet
- $\Delta$ : output alphabet
- $\delta: Q \times \Sigma \rightarrow Q$  (transition function)
- $\lambda: Q \rightarrow \Delta$  (output function)
- $q_0 \in Q$ : initial state

Output is produced on entry to the state.

### Moore Machine – Example

Design a Moore machine to generate 1's complement of a given binary number.



### 1's Complement in Moore Machine

States	Output	Input "0"	Input "1"
q0	0	q2	q1
q1	0	q2	q1
q2	1	q2	q1

### Mealy Machine

- A finite state machine where output is associated with *transitions*.

#### Definition:

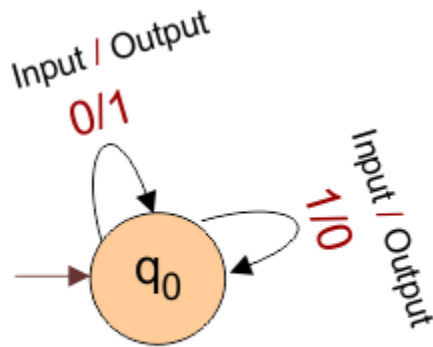
A 6-tuple  $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ , where:

- $Q$ : set of states
- $\Sigma$ : input alphabet
- $\Delta$ : output alphabet
- $\delta: Q \times \Sigma \rightarrow Q$
- $\lambda: Q \times \Sigma \rightarrow \Delta$

Output is produced on reading the input symbol.

## Mealy Machine – Example

Design a Mealy machine to generate the first complement of any given binary input.



### 1st Complement in Mealy Machine

Input	1	1	1	0	0
State	q0	q0	q0	q0	q0
Output	0	0	0	1	1

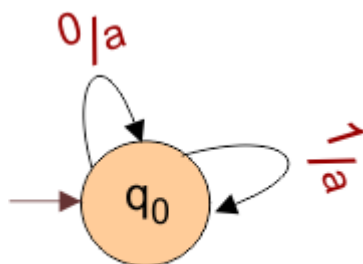
## Conversion Between Moore and Mealy

### Mealy → Moore:

- For each state/input/output pair, create a new Moore state.
- Adjust transitions to match new states and outputs.

### Conversion: Mealy → Moore Example

Consider the following Mealy Machine



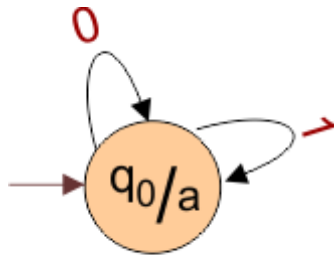
### Mealy Machine

As in the above Mealy Machine,

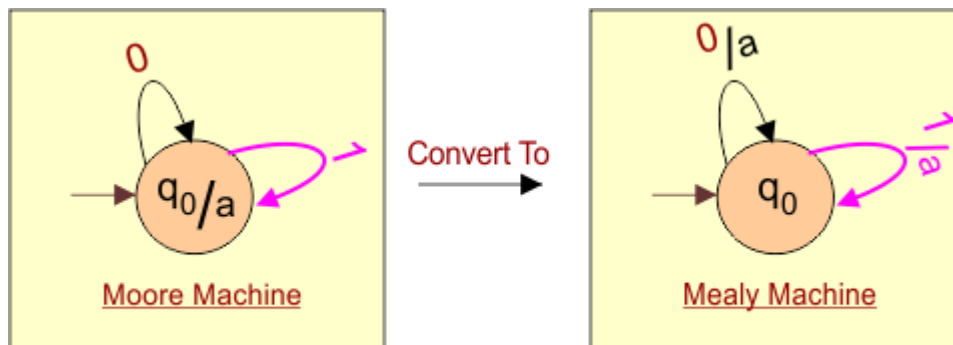
- q0 is the start state, (0,1) are inputs, and “a” is the output.
- Every entering input in the state q0 having the similar output “a”.
- So, simply cut the output “a” over the arrow and place it along with the state “q0”.



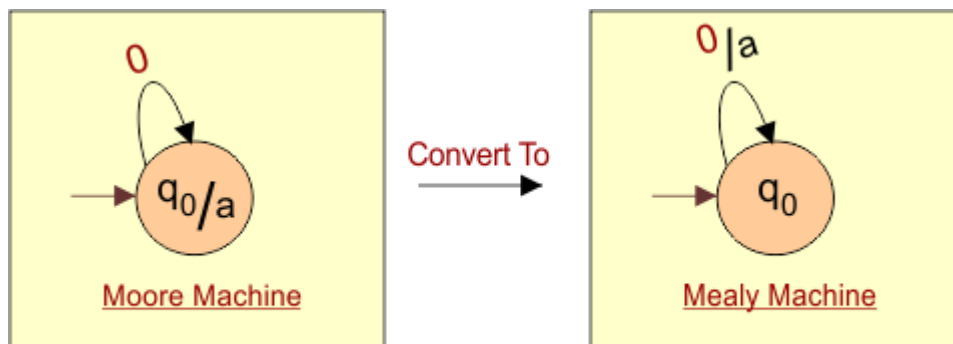
Moore → Mealy:



Moore Machine

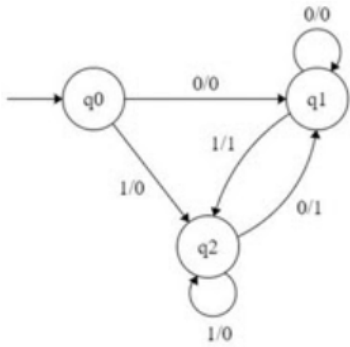
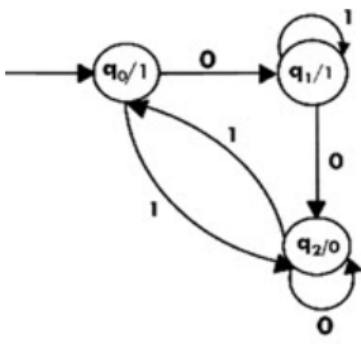


At  $q_0$  For Input "1"



At  $q_0$  For Input "0"

### Moore vs Mealy – Comparison Table

Mealy Machine	Moore Machine
Output depends on present state as well as present input.	Output depends only upon the present state.
If input changes, output also changes.	If input changes, output does not change.
Less number of states are required.	More states are required.
Asynchronous output generation.	Synchronous output and state generation.
Output is placed on transition.	Output is placed on state.
It is difficult to design.	Easy to design.
 <p>A Mealy Machine state transition diagram with three states: q0, q1, and q2. q0 is the start state. Transitions are labeled with input/output pairs: q0 to q1 (0/0), q0 to q2 (1/0), q1 to q1 (0/0), q1 to q2 (0/1), q2 to q1 (1/1), and q2 to q2 (1/0).</p>	 <p>A Moore Machine state transition diagram with three states: q0/1, q1/1, and q2/0. q0/1 is the start state. Transitions are labeled with input only: q0/1 to q1/1 (0), q1/1 to q2/0 (0), q2/0 to q0/1 (1), and q2/0 to q1/1 (1). Each state has a self-loop labeled with the opposite input (1 for q0/1 and q1/1, 0 for q2/0).</p>

## 2.4 Non-Deterministic Finite Automata (NFA) and Equivalence with DFA

### Non-Deterministic Finite Automata (NFA)

NFAs allow multiple or zero transitions for a given input, including  $\epsilon$ -transitions.

#### Definition:

An NFA is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$ , where:

- $\delta: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$  (can go to zero or more states)

#### Acceptance:

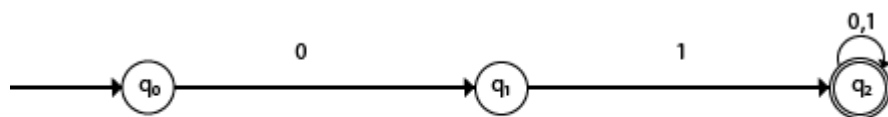
A string is accepted if at least one computation path ends in a final state.

#### Equivalence with DFA

- **Subset Construction Algorithm** (also called powerset construction) is used to convert an NFA to an equivalent DFA.

**Example:**

NFA with  $\Sigma = \{0, 1\}$  accepts all strings with 01.

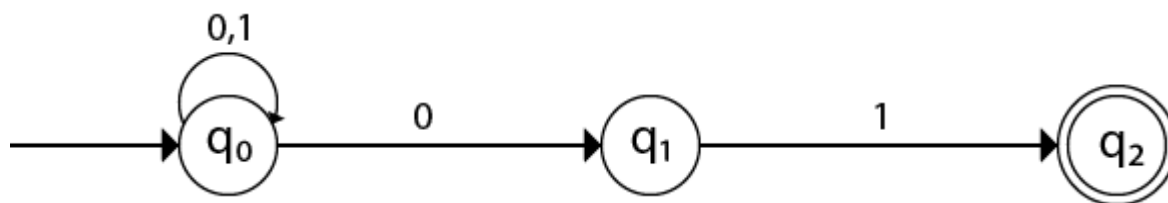


**Fig: NFA**

Present State	Next state for Input 0	Next State of Input 1
→q0	q1	ε
q1	ε	q2
*q2	q2	q2

Design an NFA with  $\Sigma = \{0, 1\}$  accepts all string ending with 01.

Anything either 0 or 1

      0   1


**Key Point:**

NFAs and DFAs recognize the **same class of languages** (regular languages), but NFAs can be exponentially more succinct.

## 2.5 Regular Grammars and Equivalence with Finite Automata

### Regular Grammar

A formal grammar with production rules that generate regular languages.

#### Types:

- **Right-linear:**  $A \rightarrow aB$  or  $A \rightarrow a$  or  $A \rightarrow \epsilon$
- **Left-linear:**  $A \rightarrow Ba$  or  $A \rightarrow a$  or  $A \rightarrow \epsilon$

### Equivalence with FA

- Every regular grammar can be converted to a finite automaton, and vice versa.
- The direction:
  - **Grammar  $\rightarrow$  FA:** Create states for variables, transitions for productions.
  - **FA  $\rightarrow$  Grammar:** States become variables, transitions become productions.

### Regular Grammar $\rightarrow$ Finite Automaton Example

#### Eliminating Epsilon Productions: A Step-by-Step Approach:

$S \rightarrow a, aA \mid bB$

$A \rightarrow aA \mid aS$

$B \rightarrow cS$

$S \rightarrow \epsilon$

$B \rightarrow \epsilon$

#### Step 1: Identifying Epsilon Productions

First, we identify the epsilon productions in our grammar. In this case, they are –

$S \rightarrow \epsilon$

$B \rightarrow \epsilon$

#### Step 2: Generating Non-Epsilon Productions

We list all the productions that do not involve epsilon. These are the productions that form the basis of our epsilon-free grammar –

$S \rightarrow a, aA \mid bB$

$A \rightarrow aA \mid aS$

$B \rightarrow cS$

### Step 3: Replacing Epsilon Productions

Now, we systematically replace all occurrences of non-terminals with epsilon productions in the right-hand sides of our productions.

In ' $A \rightarrow aA \mid aS$ ', replacing ' $S$ ' with ' $\epsilon$ ' yields ' $A \rightarrow aA \mid \epsilon$ '. Since ' $\epsilon$ ' is the empty string, we can simplify this to ' $A \rightarrow aA$ '.

In ' $B \rightarrow cS$ ', replacing ' $S$ ' with ' $\epsilon$ ' yields ' $B \rightarrow c$ '.

In ' $S \rightarrow aA \mid bB$ ', replacing ' $B$ ' with ' $\epsilon$ ' yields ' $S \rightarrow aA \mid b$ '.

The final productions will be like,

$S \rightarrow a, aA \mid b$

$A \rightarrow aA \mid aS \mid a$

$B \rightarrow cS \mid c$

### Step 4: Handling the Start Symbol

The start symbol ' $S$ ' has an epsilon production. This means that the start symbol can derive the empty string, which is often acceptable in finite automata.

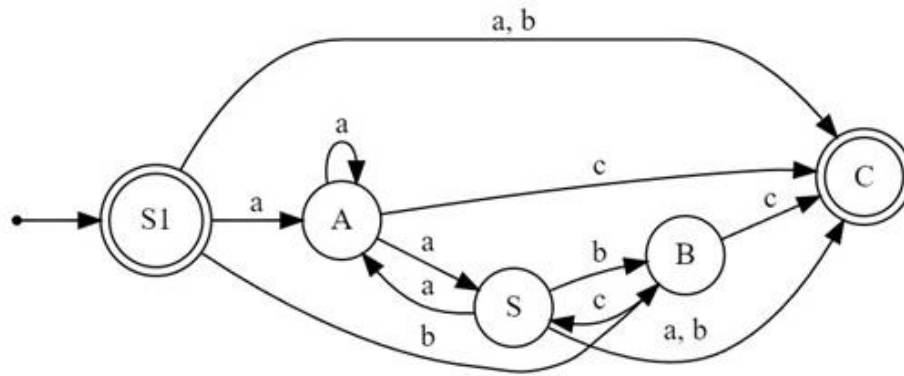
However, if the start symbol has an epsilon production, we need to add a new start symbol (' $S_1$ ' in our example) that inherits all the productions of the original start symbol, including the epsilon production. This new start symbol ensures that the empty string can be accepted by the automata.

$S_1 \rightarrow a \mid aA \mid b \mid bB \mid \epsilon$

$S \rightarrow a, aA \mid b$

$A \rightarrow aA \mid aS \mid a$

$B \rightarrow cS \mid c$



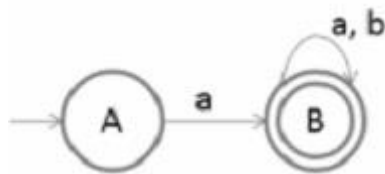
### Finite Automaton → Regular Grammar

#### Steps:

For each transition from state A to B on input a, add production  $A \rightarrow aB$

For each final state F, add production  $F \rightarrow \epsilon$

Let's consider a Finite automaton (FA) as given below –



Pick the start state A and output is on symbol 'a' going to state B

$$A \rightarrow aB$$

Now we will pick state B and then we will go on each output

$$\text{i.e } B \rightarrow aB$$

$$B \rightarrow bB$$

$$B \rightarrow \epsilon$$

Therefore,

Final grammar is as follows –

$$A \rightarrow aB$$

$$B \rightarrow aB/bB/\epsilon$$

## 2.6 Properties of Regular Languages and Pumping

### Lemma

#### Properties (Closure)

Regular languages are closed under:

- Union
- Concatenation
- Kleene Star
- Intersection
- Complement
- Difference
- Reversal

#### Pumping Lemma

Used to prove that a language is **not regular**.

#### Statement:

For any regular language  $L$ , there exists a constant  $p$  (pumping length), such that any string  $s \in L$  with  $|s| \geq p$  can be divided into:

- $s = xyz$ 
  - $|xy| \leq p$
  - $|y| \geq 1$
  - $\forall i \geq 0, xy^iz \in L$

#### Proof Technique:

- Assume  $L$  is regular.
- Let  $s \in L$ ,  $|s| \geq p$ .
- Show no division  $xyz$  satisfies the conditions  $\rightarrow$  contradiction.

## 2.7 Minimization of Finite Automata

#### Objective:

To reduce the number of states in a DFA while preserving its language.

#### Steps:

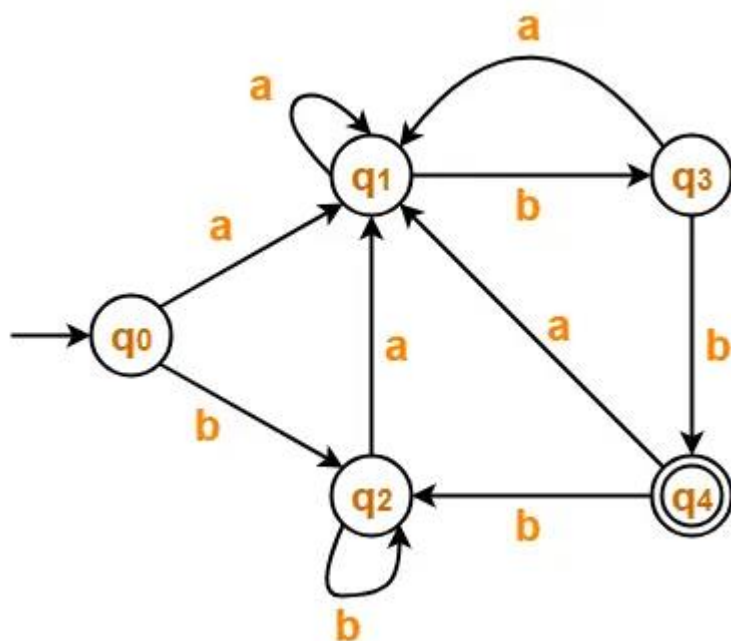
##### 1. Remove Unreachable States

- Perform DFS/BFS from the initial state.

## 2. Distinguish States Using Partitioning:

- Initially partition states into two sets: final and non-final.
- Iteratively refine the partition by distinguishing states with different future behavior.

**Example:**



**Step-01:**

The given DFA contains no dead states and inaccessible states.

**Step-02:**

Draw a state transition table-

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



**Step-03:**

Now using Equivalence Theorem, we have-

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

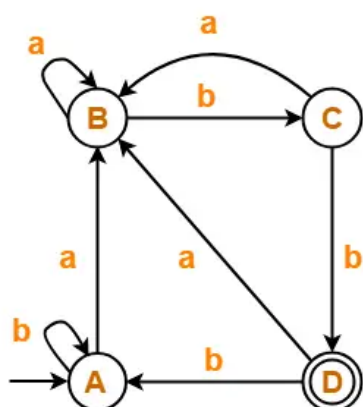
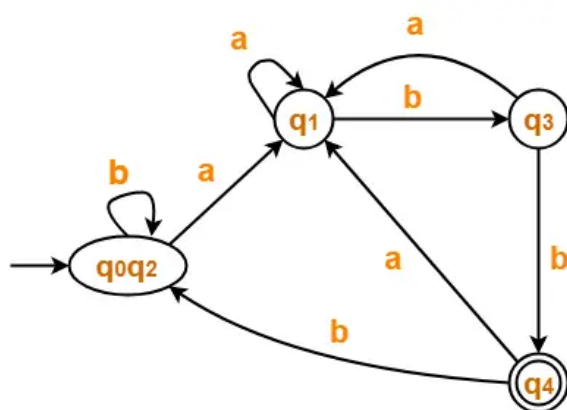
$$P_1 = \{q_0, q_1, q_2\} \{q_3\} \{q_4\}$$

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

$$P_3 = \{q_0, q_2\} \{q_1\} \{q_3\} \{q_4\}$$

Since  $P_3 = P_2$ , so we stop.

From  $P_3$ , we infer that states  $q_0$  and  $q_2$  are equivalent and can be merged together.



**Minimal DFA**

## References:

1. Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2006). *Introduction to Automata Theory, Languages, and Computation* (3rd ed.). Pearson Education. — Chapter 3: Regular Expressions and Languages.
2. Tutorialspoint. (n.d.). *Formal Grammar and Language*. Retrieved from [https://www.tutorialspoint.com/automata\\_theory/formal\\_grammar.htm](https://www.tutorialspoint.com/automata_theory/formal_grammar.htm)
3. GeeksforGeeks. (n.d.). Chomsky Hierarchy in Theory of Computation. <https://www.geeksforgeeks.org/chomsky-hierarchy/>
4. Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2006). *Introduction to Automata Theory, Languages, and Computation*. — Section on finite-state transducers (Moore and Mealy machines).
5. Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2006). — Chapter 2: NFAs,  $\epsilon$ -NFAs and DFA Equivalence.
6. Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2006). — Chapter 3: Regular Grammars and their Correspondence with Automata.
7. Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2006). — Chapter 4: DFA Minimization Algorithms (Table-Filling and Hopcroft's Algorithm).

**Parul<sup>®</sup> University** | **NAAC** **A++**  
Vadodara, Gujarat | **GRADE**



      
<https://paruluniversity.ac.in/>