

THEORY OF COMPUTATION

Chapter 2

Finite Automata

- 2.1** Automaton, Finite Automaton, Block Diagram, Transition System, Properties of Transition Functions, Acceptability of a String
- 2.2** DFA and NDFA, Equivalence of DFA and NDFA, Theorem on Equivalence (with Applications)
- 2.3** Mealy and Moore Machines, Conversion Procedures between them (with Applications)

Practice: 15 MCQs with Answer Key and 10 Broad Questions

Notes in paragraph, list and table format with diagrams, prepared for B.E. / B.Tech / Diploma engineering students (Computer Science, IT and related branches).

Poly Notes Hub

Chapter 2: Finite Automata

Overview

A **finite automaton (FA)** is the simplest model of computation. It has a fixed, finite number of states and **no extra memory**: all it can remember about the input read so far is the state it is currently in. Despite this limitation, finite automata are used everywhere: in the lexical analyser of a compiler, in pattern-matching tools, in digital circuits and in the control logic of devices such as vending machines and traffic lights. In this chapter we first fix the basic vocabulary (states, transition function, acceptance), then compare **deterministic** and **non-deterministic** automata and prove that they have the same power, and finally study automata that **produce output**: the Moore and Mealy machines.

Classification of Finite Automata

Category	Machine	Output	Used for
Without output (acceptors)	DFA, NDFA (NFA), NFA with ϵ -moves	Accept / reject the whole string	Recognising regular languages
With output (transducers)	Moore machine, Mealy machine	A string of output symbols	Circuit design, converters, controllers

Chapter Map

Section	Topics	Key idea
2.1	Automaton, finite automaton, block diagram, transition system, properties of δ , acceptability	A string is accepted if the machine ends in a final state
2.2	DFA, NDFA, equivalence theorem, subset construction, applications	Non-determinism adds convenience, not power
2.3	Moore and Mealy machines, conversion in both directions, applications	Output depends on state (Moore) or on state and input (Mealy)

2.1 Finite Automaton and Acceptability of Strings

2.1.1 Definition of an Automaton

An **automaton** is an abstract machine that reads a string over an input alphabet one symbol at a time and moves from one *configuration* to another according to fixed rules. At the end it decides whether the string belongs to the language it is built to recognise. The word *automaton* means "self-acting": once the input is given, the machine works on its own with no outside help.

Definition: Automaton

An automaton is a mathematical model of a machine with (i) a set of **states**, (ii) an **input alphabet**, (iii) a **transition rule** that gives the next configuration from the current one, (iv) a **start** configuration and (v) a set of **accepting** configurations. Different kinds of automata differ in the amount of memory available to them.

Property of an automaton	Meaning
Input	Finite string over the input alphabet Σ , written on the input tape
States	The machine is always in exactly one state of a set Q
Transition	Move from one state to another after reading a symbol
Deterministic / non-deterministic	One possible move versus several possible moves
Acceptor / transducer	Gives yes/no answer versus gives an output string

2.1.2 Definition of a Finite Automaton

Definition: Finite Automaton

A **finite automaton** is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where:

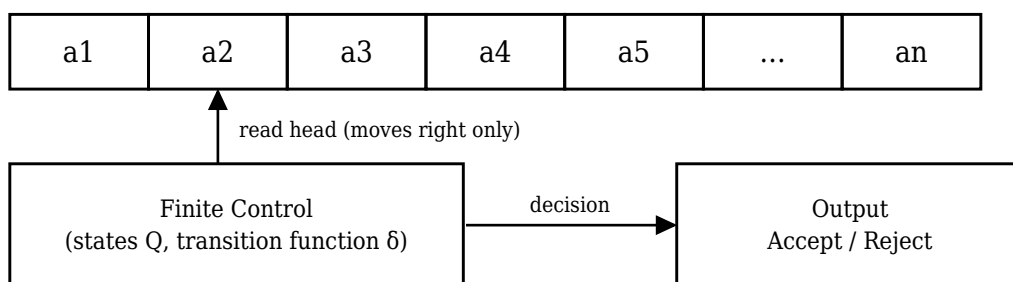
- Q is a finite non-empty set of **states**;
- Σ is a finite non-empty set of **input symbols** (the input alphabet);
- δ is the **transition function** (for a DFA, $\delta : Q \times \Sigma \rightarrow Q$);
- $q_0 \in Q$ is the **initial state**;
- $F \subseteq Q$ is the set of **final (accepting) states**.

The word **finite** refers to the finite number of states. Because the machine can only distinguish finitely many situations, it cannot count without a bound. For example, a finite automaton cannot recognise $\{a^n b^n : n \geq 0\}$, since it would need a different state for every possible value of n .

2.1.3 Block Diagram of a Finite Automaton

A finite automaton is pictured as a box (the *finite control*) connected to an input tape by a read head. The head starts on the leftmost symbol and moves one cell to the right after each move. It never writes on the tape and never moves left, so each input symbol is examined exactly once.

Input tape (finite string, read left to right)



No auxiliary memory: the only memory is the current state

Figure 2.1: Block diagram of a finite automaton

Part	Function
Input tape	Divided into cells; each cell holds one input symbol $a_i \in \Sigma$. The string ends at the last cell.
Read head	Reads the symbol under it, then moves one cell right. Read-only.
Finite control	Stores the current state and applies δ to (current state, input symbol) to find the next state.
Output	After the last symbol: accept if the state is in F , otherwise reject.

2.1.4 Transition System

A finite automaton is usually described by a **transition system**, which is a directed graph whose vertices are the states and whose edges are labelled with input symbols. In its most general form (used in many textbooks) it allows a *set* of initial states and edges labelled by strings.

Definition: Transition System

A transition system is a 5-tuple $(Q, \Sigma, \delta, Q_0, F)$ where Q is a finite set of states, Σ a finite input alphabet, δ a finite set of transitions (each from a state to a state, labelled by a string in Σ^*), $Q_0 \subseteq Q$ a non-empty set of initial states and $F \subseteq Q$ the set of final states. A DFA is the special case with exactly one initial state, edges labelled by single symbols, and exactly one outgoing edge for every state and symbol.

Representations of a transition system

Representation	Description	Convention
Transition diagram (graph)	Circles for states, labelled arrows for transitions	Arrow from nowhere = start state; double circle = final state; loop = self-transition
Transition table	Rows are states, columns are input symbols, entries are next states	$\rightarrow$ marks the start state; * marks a final state
Transition function	Formula $\delta(q, a) = p$	Lists all the pairs (q, a)

2.1.5 Properties of Transition Functions

The one-step function $\delta(q, a)$ handles a single symbol. To process a whole string we extend it to δ^* (often simply written δ again) defined recursively on the length of the string.

Extended transition function δ^*

- (i) $\delta^*(q, \epsilon) = q$ (reading nothing leaves the state unchanged)
- (ii) $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$ for every string w and symbol a

No.	Property	Explanation
1	$\delta(q, \epsilon) = q$	No input, no move
2	$\delta^*(q, a) = \delta(q, a)$	On a single symbol the extended and the basic function agree: $\delta^*(q, \epsilon a) = \delta(\delta^*(q, \epsilon), a) = \delta(q, a)$
3	$\delta^*(q, xa) = \delta(\delta^*(q, x), a)$	Definition: process x first, then the last symbol a
4	$\delta^*(q, xy) = \delta^*(\delta^*(q, x), y)$	Processing xy equals processing x and then y from the state reached
5	δ is a total function (DFA)	Every pair (q, a) has exactly one next state
6	$\delta(P, a) = \cup \delta(p, a), p \in P$ (NFA)	For a set of states P the next set is the union of the individual results

Proof of Property 4: $\delta^*(q, xy) = \delta^*(\delta^*(q, x), y)$

Use induction on $|y|$, keeping x and q fixed.

- **Basis ($|y| = 0, y = \epsilon$):** $\delta^*(q, x\epsilon) = \delta^*(q, x)$. Also $\delta^*(\delta^*(q, x), \epsilon) = \delta^*(q, x)$ by rule (i). Both sides are equal.
- **Induction step:** let $y = wa$ and assume the result for w , i.e. $\delta^*(q, xw) = \delta^*(\delta^*(q, x), w)$. Then $\delta^*(q, xwa) = \delta(\delta^*(q, xw), a)$ [rule (ii)] = $\delta(\delta^*(\delta^*(q, x), w), a)$ [hypothesis] = $\delta^*(\delta^*(q, x), wa)$ [rule (ii)]. This is the claim for $y = wa$. (Hence proved.)

2.1.6 Acceptability of a String by a Finite Automaton

Definition: Acceptance and Language

A string $w \in \Sigma^*$ is **accepted** by $M = (Q, \Sigma, \delta, q_0, F)$ if $\delta^*(q_0, w) \in F$, that is, starting in q_0 and reading w the machine halts in a final state. Otherwise w is **rejected**. The **language accepted** by M is $L(M) = \{ w \in \Sigma^* : \delta^*(q_0, w) \in F \}$.

- The machine reads the *entire* string; it cannot stop early (unlike a program that can return in the middle).
- ϵ is accepted exactly when the initial state is also a final state ($q_0 \in F$).
- A state from which no final state can be reached is a **dead (trap) state**; strings entering it are rejected for ever.
- Two automata are **equivalent** if they accept the same language.

Example: DFA that accepts binary numbers divisible by 3

Reading a binary number from the left, if the number read so far is v and the next bit is b , the new number is $2v + b$. So only the remainder of v modulo 3 matters. States r_0, r_1, r_2 store the remainder; the new remainder is $(2i + b) \bmod 3$. Start and final state is r_0 .

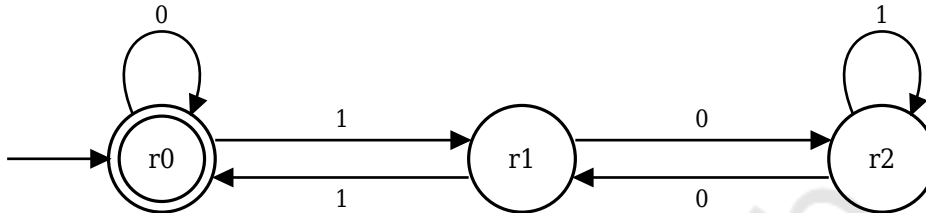


Figure 2.2: DFA accepting binary strings whose value is divisible by 3

State	On input 0	On input 1
$\rightarrow^* r_0$	r_0	r_1
r_1	r_2	r_0
r_2	r_1	r_2

Input w	Value	Path of states	Last state	Result
110	6	$r_0 \xrightarrow{1} r_1 \xrightarrow{1} r_0 \xrightarrow{0} r_0$	r_0	Accepted
101	5	$r_0 \xrightarrow{1} r_1 \xrightarrow{0} r_2 \xrightarrow{1} r_2$	r_2	Rejected
1001	9	$r_0 \xrightarrow{1} r_1 \xrightarrow{0} r_2 \xrightarrow{0} r_1 \xrightarrow{1} r_0$	r_0	Accepted
ϵ	0	r_0	r_0	Accepted

Steps for designing a DFA

- Decide **what must be remembered** about the part of the input already read (a remainder, the last symbol, a count up to a limit).
- Create one **state for each distinct situation** and mark the start state.
- Write the **transition for every state and every symbol**; add a dead state for impossible continuations.
- Mark as **final** the states that correspond to "string belongs to the language".
- Test** with short accepted and rejected strings, including ϵ .

Language over $\{0, 1\}$ (unless stated)	What the states remember	States
Even number of 1s (parity)	Parity of 1s read so far	2
Strings ending with 01	Progress towards the suffix 01	3
Strings containing substring 11	Seen 11 yet? last symbol 1?	3
Length is a multiple of 3	Length mod 3	3

2.2 DFA and N DFA, and their Equivalence

2.2.1 Deterministic Finite Automaton (DFA)

Definition: DFA

A DFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ with $\delta : Q \times \Sigma \rightarrow Q$. For every state and every input symbol there is **exactly one** next state.

- The run of a DFA on a string is a **single, uniquely determined path** of states.
- The transition table has exactly one entry in every cell.
- A DFA is easy to simulate on a computer: one table look-up per input symbol, so time is proportional to the length of the input.

2.2.2 Non-deterministic Finite Automaton (N DFA / NFA)

Definition: N DFA

An N DFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ with $\delta : Q \times \Sigma \rightarrow 2^Q$ (2^Q is the power set of Q). The value $\delta(q, a)$ is a **set** of states, possibly empty or containing several states.

In an N DFA, from a state on a symbol the machine may have **zero, one or several** choices. We say it "guesses" the right path. The extended function is $\delta^*(q, \epsilon) = \{q\}$ and $\delta^*(q, wa) = \cup \{ \delta(p, a) : p \in \delta^*(q, w) \}$. A string w is accepted if $\delta^*(q_0, w) \cap F \neq \emptyset$, that is, *at least one* sequence of choices ends in a final state. The language is $L(M) = \{ w : \delta^*(q_0, w) \cap F \neq \emptyset \}$.

Example: N DFA accepting strings that end with 01

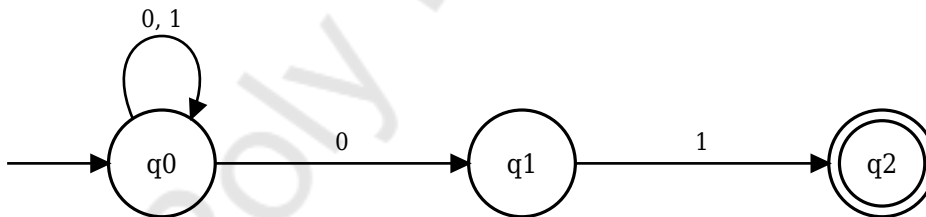


Figure 2.3: N DFA accepting binary strings ending with 01 (q_0 loops on both symbols)

State	On input 0	On input 1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
$* q_2$	$\emptyset$	$\emptyset$

Trace for 1001 (the set of possible states after each symbol): $\{q_0\} \xrightarrow{1} \{q_0\} \xrightarrow{0} \{q_0, q_1\} \xrightarrow{0} \{q_0, q_1\} \xrightarrow{1} \{q_0, q_2\}$. The final set contains $q_2 \in F$, so 1001 is **accepted**. For 110 the sets are $\{q_0\}, \{q_0\}, \{q_0, q_1\}$; no final state appears, so 110 is **rejected**.

NDFA with ϵ -moves

An ϵ -NFA additionally allows transitions on the empty string, $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$. The ϵ -closure of a state q is the set of states reachable from q using only ϵ -moves (including q itself). ϵ -moves make it easy to combine automata: for the union of two languages, add a new start state with ϵ -moves to the two old start states. ϵ -NFAs, NFAs and DFAs all accept exactly the regular languages.

2.2.3 Comparison of DFA and NDFA

Aspect	DFA	NDFA
Transition function	$\delta : Q \times \Sigma \rightarrow Q$	$\delta : Q \times \Sigma \rightarrow 2^Q$
Next state	Exactly one	Zero, one or many
Run on a string	One unique path	Many possible paths (a computation tree)
Acceptance	The single path ends in F	At least one path ends in F
Dead state	Needed for a complete table	Not needed (missing move = empty set)
Ease of design	Longer, more states	Usually shorter and more natural
Implementation	Direct and fast	Must be simulated or converted to a DFA
Power	Regular languages	Regular languages (same)

2.2.4 Equivalence of DFA and NDFA

Every DFA is trivially an NDFA (each set $\delta(q, a)$ has exactly one element). The surprising direction is the converse: an NDFA, however many choices it has, can be simulated by a DFA that keeps track of *all* states the NDFA could be in at the same time.

Theorem (Equivalence of DFA and NDFA, Rabin and Scott)

For every NDFA $N = (Q, \Sigma, \delta, q_0, F)$ there is a DFA D with $L(D) = L(N)$. Hence a language is accepted by a DFA if and only if it is accepted by an NDFA.

Proof (subset construction)

Define the DFA $D = (Q', \Sigma, \delta', q_0', F')$ as follows.

Component	Definition
States Q'	$Q' = 2^Q$: every state of D is a subset of Q
Start state q_0'	$\{q_0\}$
Transition δ'	$\delta'(P, a) = \cup \{ \delta(p, a) : p \in P \}$ for every $P \subseteq Q$ and $a \in \Sigma$
Final states F'	$F' = \{ P \subseteq Q : P \cap F \neq \emptyset \}$ (subsets containing at least one final state of N)

Claim: for every string w , $\delta'^*(\{q_0\}, w) = \delta^*(q_0, w)$. We prove it by induction on $|w|$.

- **Basis ($w = \epsilon$):** $\delta'^*(\{q_0\}, \epsilon) = \{q_0\} = \delta^*(q_0, \epsilon)$.
- **Step ($w = xa$):** $\delta'^*(\{q_0\}, xa) = \delta'(\delta'^*(\{q_0\}, x), a) = \delta'(\delta^*(q_0, x), a)$ [hypothesis] $= \cup \{ \delta(p, a) : p \in \delta^*(q_0, x) \}$ [definition of δ'] $= \delta^*(q_0, xa)$.

Therefore $w \in L(D) \Leftrightarrow \delta'^*(\{q_0\}, w) \in F' \Leftrightarrow \delta^*(q_0, w) \cap F \neq \emptyset \Leftrightarrow w \in L(N)$. (Hence proved.)

Procedure to convert an NFA into a DFA

- Start with the subset $\{q_0\}$ as the start state of the DFA.
- For the subset P just added and each symbol a , compute $\delta'(P, a)$ as the union of $\delta(p, a)$ over $p \in P$.
- If the result is a **new subset**, add it as a new state of the DFA. The empty subset $\emptyset$ becomes the dead state.
- Repeat until no new subsets appear. (Only **reachable** subsets are built, which keeps the DFA small.)
- Mark as **final** every subset that contains at least one final state of the NFA.

Worked example: convert the NFA of Figure 2.3 into a DFA

Subset (DFA state)	On 0	On 1	Final?
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$	No
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$	No
$* \{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$	Yes (contains q_2)

Computation: $\delta'(\{q_0, q_1\}, 1) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$; $\delta'(\{q_0, q_2\}, 0) = \{q_0, q_1\} \cup \emptyset$. Only 3 of the $2^3 = 8$ possible subsets are reachable.

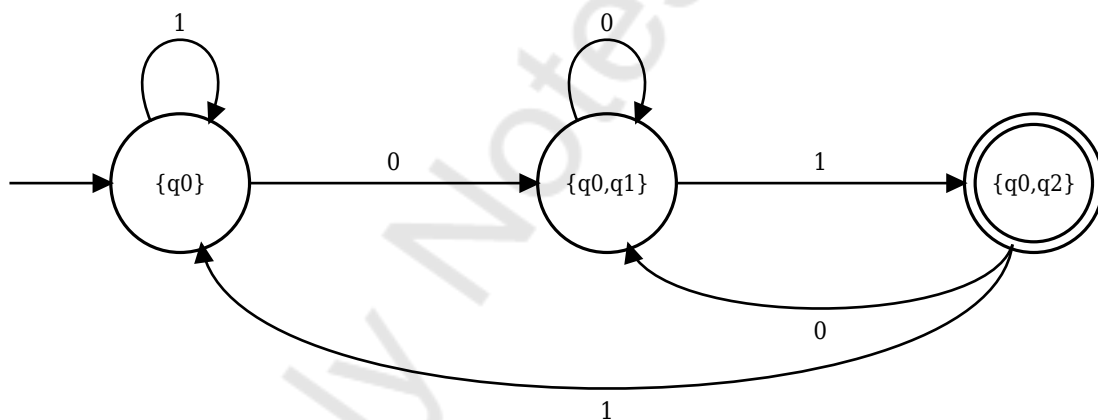


Figure 2.4: DFA obtained from the NFA by the subset construction

Remarks

- In the worst case the DFA needs 2^n states for an n -state NFA; some languages really need this many (for example "the n -th symbol from the right end is 1").
- The theorem shows that non-determinism does **not** enlarge the class of languages of finite automata; it only makes machines easier to write and sometimes exponentially smaller.
- For pushdown automata the same statement is **false**: non-deterministic PDAs are strictly more powerful.

2.2.5 Applications of DFA and NFA

Application	How automata are used
Parity and error checking	A 2-state DFA tracks whether the number of 1s is even or odd (parity bit check).
Arithmetic on bit strings	Divisibility tests such as "divisible by 3" (Figure 2.2) with a few states.
Lexical analysis	Each token class (identifier, number, keyword) is a regular expression converted via NFA to a DFA; the scanner runs the DFA over the source code.
Pattern matching	Search for a word or pattern in text (grep, editors, virus scanners, DNA search) by building a DFA of the pattern, so the text is scanned once.
Network protocols	Connection set-up and tear-down are finite-state systems with states such as LISTEN, ESTABLISHED, CLOSED.
Controllers	Vending machines, elevators, traffic lights and ATMs move between a finite number of control states.
Design of NFAs	The NDFA is used as a design tool: guess a position (for example the start of the suffix 01) and verify it, then convert to a DFA for implementation.

2.3 Mealy and Moore Machines

2.3.1 Finite Automata with Output

A DFA only answers yes or no. Many practical devices must **produce a string of output symbols** as they read the input: an adder, a code converter, a controller. A finite automaton with output is called a **transducer**. There are two classical kinds, named after E. F. Moore (1956) and G. H. Mealy (1955). They differ in where the output is attached.

2.3.2 Moore Machine

Definition: Moore Machine

A Moore machine is a 6-tuple $\mathbf{M} = (\mathbf{Q}, \Sigma, \Delta, \delta, \lambda, q_0)$ where $\mathbf{Q}$, Σ , δ and q_0 are as in a DFA, Δ is the finite **output alphabet** and $\lambda : \mathbf{Q} \rightarrow \Delta$ is the **output function**: each state has a fixed output symbol. There are no final states.

- The output depends **only on the current state**; it is written inside the circle as **q / output**.
- On input $a_1 a_2 \dots a_n$ the machine goes through states $q_0, q_1, \dots, q_n$ and emits $\lambda(q_0) \lambda(q_1) \dots \lambda(q_n)$: an output of length $n + 1$. The first output $\lambda(q_0)$ is produced before any input is read.

Example: Moore machine giving the remainder modulo 3 of a binary number

Use the states of Figure 2.2, but now every state outputs its own remainder.

State	On 0	On 1	Output λ
$\rightarrow r_0$	r_0	r_1	0
r_1	r_2	r_0	1
r_2	r_1	r_2	2

For the input 1010 the states are r_0, r_1, r_2, r_2, r_1 and the output is **0 1 2 2 1**. The last symbol 1 is the remainder of 10 (decimal) divided by 3.

2.3.3 Mealy Machine

Definition: Mealy Machine

A Mealy machine is a 6-tuple $\mathbf{M} = (\mathbf{Q}, \Sigma, \Delta, \delta, \lambda, q_0)$ where the components are as for a Moore machine except that the output function is $\lambda : \mathbf{Q} \times \Sigma \rightarrow \Delta$: the output is attached to the **transition**, i.e. it depends on the current state *and* the current input symbol.

- In the diagram an edge is labelled **input / output**, e.g. 1/0.
- For input of length n a Mealy machine produces output of length exactly n .

Example: Mealy machine that outputs 1 whenever the current and the previous input symbol are both 1

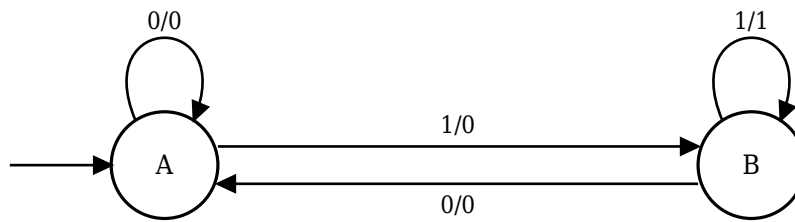


Figure 2.5: Mealy machine: output 1 when two consecutive 1s are read

State	Input 0: next / output	Input 1: next / output
→ A (last symbol not 1)	A / 0	B / 0
B (last symbol was 1)	A / 0	B / 1

Trace for 0111: A -0/0→ A -1/0→ B -1/1→ B -1/1→ B. Output = **0011** (4 symbols for 4 inputs).

2.3.4 Moore versus Mealy

Aspect	Moore machine	Mealy machine
Output function	$\lambda : Q \rightarrow \Delta$	$\lambda : Q \times \Sigma \rightarrow \Delta$
Output depends on	Current state only	Current state and current input
Output attached to	States (q / output)	Transitions (input / output)
Output length for n inputs	n + 1 (includes the start-state output)	n
Reaction to input	Output appears after the state change (one step later)	Output appears immediately with the input
Number of states	Usually more	Usually fewer
Typical use	Circuits whose outputs are registered and glitch-free	Fast, compact controllers

2.3.5 Equivalence of Moore and Mealy Machines

A Moore machine M_1 and a Mealy machine M_2 are called **equivalent** if, for every input string w , the Moore machine outputs **b followed by $T_2(w)$** where $b = \lambda(q_0)$ is its start-state output and $T_2(w)$ is the Mealy output. Because the Moore machine's first symbol does not depend on the input, it is ignored in the comparison. With this understanding, **every Moore machine has an equivalent Mealy machine and vice versa**, so the two models have the same power.

2.3.6 Procedure to Transform a Mealy Machine into a Moore Machine

- For each state q_i , collect the **set of output symbols** on all **incoming** transitions of q_i .
- If q_i has **m different incoming outputs**, split it into m states (q_i, o) one for each output o . If q_i has no incoming transitions, keep a single copy with an arbitrary output (usually 0).
- The Moore output of the state (q_i, o) is **o** .

- For every Mealy transition $q_i \xrightarrow{a/o} q_j$, in the Moore machine draw an edge on a from **every** copy of q_i to the copy (q_j, o) .
- Choose as start state the copy of q_0 whose output is the start output (take 0 if free). In general the number of Moore states is at most $|Q| \times |\Delta|$.

Example 1: convert the Mealy machine of Figure 2.5 into a Moore machine

Incoming outputs: state A receives only output 0 (from A on 0 and from B on 0), so A is not split. State B receives output 0 (from A on 1) and output 1 (from B on 1), so B is split into B0 (output 0) and B1 (output 1).

Moore state	On 0	On 1	Output
→ A	A	B0	0
B0	A	B1	0
B1	A	B1	1

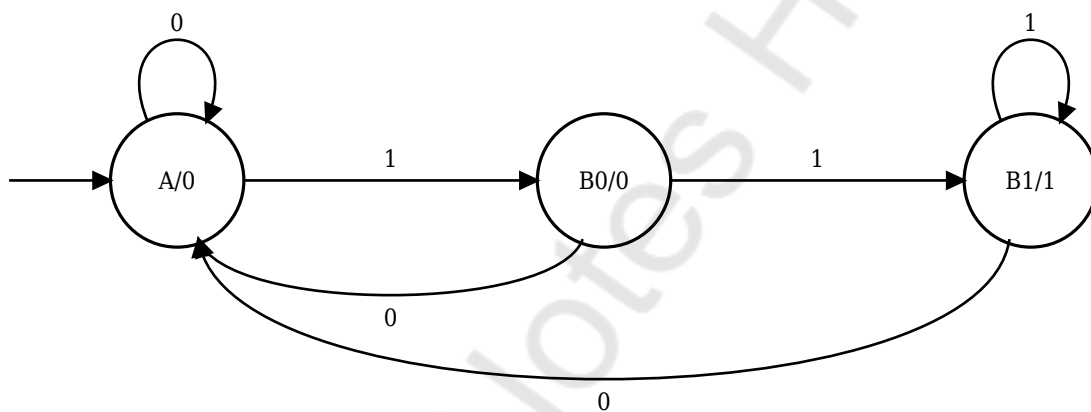


Figure 2.6: Moore machine equivalent to the Mealy machine of Figure 2.5

Check: for 0111 the Moore machine visits A, A, B0, B1, B1 and outputs 0 0 0 1 1. Dropping the first output gives 0 0 1 1, the same as the Mealy output.

Example 2: a Mealy machine whose states both split

Mealy machine with states q_1 (start), q_2 : q_1 : on 0 $\rightarrow q_1/1$, on 1 $\rightarrow q_2/0$; q_2 : on 0 $\rightarrow q_1/0$, on 1 $\rightarrow q_2/1$.

Mealy state	Incoming outputs	Moore states formed
q_1	1 (from q_1 on 0), 0 (from q_2 on 0)	$q_1,0$ and $q_1,1$
q_2	0 (from q_1 on 1), 1 (from q_2 on 1)	$q_2,0$ and $q_2,1$

Moore state	On 0	On 1	Output
→ ($q_1,0$)	($q_1,1$)	($q_2,0$)	0
($q_1,1$)	($q_1,1$)	($q_2,0$)	1
($q_2,0$)	($q_1,0$)	($q_2,1$)	0
($q_2,1$)	($q_1,0$)	($q_2,1$)	1

The number of states grew from 2 to $4 = |Q| \times |\Delta|$, the upper limit. Check with input 10: Mealy output 0 0; Moore path $(q_1, 0), (q_2, 0), (q_1, 0)$ gives output 0 0 0, which is 0 followed by 0 0.

2.3.7 Procedure to Transform a Moore Machine into a Mealy Machine

- Keep the **same states**, the same input alphabet and the same transitions δ .
- For every transition $q \xrightarrow{a} p$ define the Mealy output $\lambda'(q, a) = \lambda(p)$: the output of the *target* state.
- The output $\lambda(q_0)$ of the Moore start state is simply **discarded** (the Mealy machine produces no output before the first input).
- Optionally **merge equivalent states** (states with identical rows in the table) to obtain a smaller machine.

Example: convert the Moore machine of Figure 2.6 into a Mealy machine

State	On 0: next / output	On 1: next / output
$\rightarrow A$	$A / \lambda(A) = 0$	$B0 / \lambda(B0) = 0$
B0	$A / 0$	$B1 / \lambda(B1) = 1$
B1	$A / 0$	$B1 / 1$

States B0 and B1 now have identical rows ($0 \rightarrow A/0, 1 \rightarrow B1/1$), so they are equivalent and can be merged into one state B. The result is exactly the two-state Mealy machine of Figure 2.5, which confirms that the two conversions are consistent.

2.3.8 Applications of Mealy and Moore Machines

(a) Serial binary adder (Mealy machine)

Two binary numbers are fed **least significant bit first**, one pair of bits at each step. The machine needs only the **carry** as memory: state c0 (carry 0) and state c1 (carry 1). The output is the sum bit.

State	Input 00	Input 01 or 10	Input 11
$\rightarrow c0$ (carry 0)	$c0 / 0$	$c0 / 1$	$c1 / 0$
c1 (carry 1)	$c0 / 1$	$c1 / 0$	$c1 / 1$

Example 3 + 3: 011 and 011 are fed as pairs (1,1), (1,1), (0,0) (LSB first). $c0 \xrightarrow{11} c1 \xrightarrow{11} c1 \xrightarrow{00} c0$. Output bits 0, 1, 1 read from the last to the first give **110 = 6**.

(b) 2's complement generator (Mealy machine)

The rule is: copy the bits **up to and including the first 1** (from the right), then invert every later bit. Feed the number LSB first.

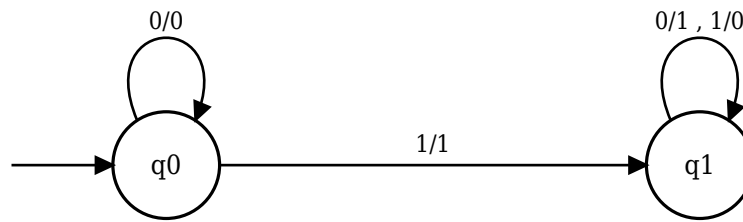


Figure 2.7: Mealy machine producing the 2's complement of a binary number (LSB first)

Example: 0110 (6) is fed as 0, 1, 1, 0: $q_0 \xrightarrow{0/0} q_0 \xrightarrow{1/1} q_1 \xrightarrow{1/0} q_1 \xrightarrow{0/1} q_1$. Output 0, 1, 0, 1 reversed is **1010**, the 4-bit 2's complement of 0110 (that is -6).

(c) Other uses

Application	Machine	Description
1's complement	Mealy (1 state)	Output the inverse of each input bit: 0/1 and 1/0 on a single state
Remainder / modulo counter	Moore	Outputs the current value of (number read) mod k; see the mod 3 example
Sequence detector	Mealy or Moore	Output 1 when a given pattern such as 101 has just been read (Figure 2.5 detects 11)
Traffic-light controller	Moore	Each state (red, green, yellow) has a fixed output lighting a lamp
Vending machine	Mealy	Input is a coin, output is the product or change, state is the amount inserted
Digital circuits	Both	Sequential circuits are Mealy or Moore machines implemented with flip-flops and logic gates

2.4 Summary of the Chapter

- A **finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ with finite states and no extra memory; it reads the input once from left to right.
- δ is extended to strings: $\delta^*(q, \varepsilon) = q$, $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$, and $\delta^*(q, xy) = \delta^*(\delta^*(q, x), y)$.
- A string w is **accepted** if $\delta^*(q_0, w) \in F$; $L(M)$ is the set of accepted strings.
- A **DFA** has exactly one move for each state and symbol; an **NFA** may have many, and accepts if at least one path reaches a final state.
- **Every NFA has an equivalent DFA** (subset construction); the DFA may need up to 2^n states.
- A **Moore** machine attaches output to states ($n + 1$ outputs); a **Mealy** machine attaches output to transitions (n outputs).
- Mealy $\rightarrow$ Moore: split each state by incoming outputs (at most $|Q| \times |\Delta|$ states). Moore $\rightarrow$ Mealy: $\lambda'(q, a) = \lambda(\delta(q, a))$.

Practice Section A: 15 Multiple Choice Questions

Choose the correct option. The answer key with short explanations is given at the end of this section.

Q1. A finite automaton has:

- (A) An unbounded stack
- (B) A finite number of states and no auxiliary memory
- (C) An infinite tape that it can write on
- (D) An infinite number of states

Q2. The transition function of a DFA is of the form:

- (A) $\delta : Q \rightarrow \Sigma$
- (B) $\delta : Q \times \Sigma \rightarrow 2^Q$
- (C) $\delta : Q \times \Sigma \rightarrow Q$
- (D) $\delta : Q \times \Delta \rightarrow \Sigma$

Q3. The transition function of an NFA is of the form:

- (A) $\delta : Q \times \Sigma \rightarrow Q$
- (B) $\delta : Q \times \Sigma \rightarrow 2^Q$
- (C) $\delta : Q \rightarrow \Delta$
- (D) $\delta : Q \times \Sigma \rightarrow \Delta$

Q4. For any state q , the extended transition function gives $\delta^*(q, \epsilon) =$

- (A) $\emptyset$
- (B) ϵ
- (C) q_0
- (D) q

Q5. A string w is accepted by a DFA $M = (Q, \Sigma, \delta, q_0, F)$ if:

- (A) $\delta^*(q_0, w) \in F$
- (B) $\delta^*(q_0, w) = q_0$
- (C) $\delta^*(F, w) = q_0$
- (D) $\delta^*(q_0, w) \notin F$

Q6. Which equality is always true for the extended transition function?

- (A) $\delta^*(q, xy) = \delta^*(q, x) \delta^*(q, y)$
- (B) $\delta^*(q, xy) = \delta^*(\delta^*(q, y), x)$
- (C) $\delta^*(q, xy) = \delta^*(\delta^*(q, x), y)$
- (D) $\delta^*(q, xy) = \delta(q, x) \cup \delta(q, y)$

Q7. An NFA with n states is converted to an equivalent DFA by the subset construction. The DFA has at most:

- (A) n states
- (B) $2n$ states
- (C) n^2 states
- (D) 2^n states

- Q8.** Which statement about DFA and NDFA is correct?
- (A) NDFA accepts more languages than DFA
 - (B) DFA accepts more languages than NDFA
 - (C) Both accept exactly the regular languages
 - (D) They accept disjoint classes of languages
- Q9.** An NDFA accepts a string if:
- (A) All possible paths end in a final state
 - (B) At least one path ends in a final state
 - (C) Exactly one path exists
 - (D) No path ends in a final state
- Q10.** In the DFA for binary numbers divisible by 3 (start state r_0 , $\delta(r_i, b) = (2i + b) \bmod 3$), the state reached after reading 1010 is:
- (A) r_0
 - (B) r_2
 - (C) No state (dead)
 - (D) r_1
- Q11.** The output of a Moore machine depends on:
- (A) Current state and current input
 - (B) Current input only
 - (C) Current state only
 - (D) Previous output only
- Q12.** The output of a Mealy machine depends on:
- (A) Current state and current input
 - (B) Current state only
 - (C) Number of states
 - (D) Final state only
- Q13.** For an input string of length n , a Moore machine produces an output string of length:
- (A) $n - 1$
 - (B) n
 - (C) $2n$
 - (D) $n + 1$
- Q14.** When a Mealy machine with $|Q|$ states and output alphabet Δ is converted to a Moore machine, the number of states is at most:
- (A) $|Q| + |\Delta|$
 - (B) $|Q| \times |\Delta|$
 - (C) $2^{|Q|}$
 - (D) $|Q|$

- Q15.** In the conversion of a Moore machine to a Mealy machine, the Mealy output on a transition $q \rightarrow p$ is:
- (A) $\lambda(q)$
 - (B) $\lambda(q) + \lambda(p)$
 - (C) $\lambda(p)$
 - (D) Always 0

Answer Key

Q	Ans	Explanation
1	B	FA has finitely many states and its only memory is the current state.
2	C	A DFA gives exactly one next state for each (state, symbol) pair.
3	B	The value is a subset of Q (an element of the power set).
4	D	Reading the empty string leaves the machine in the same state.
5	A	Acceptance means ending in a final state after reading the whole string.
6	C	Process x first, then y starting from the state reached (proved by induction on $ y $).
7	D	Each DFA state is a subset of the n NFA states, giving up to 2^n subsets.
8	C	The equivalence theorem: both models accept the regular languages.
9	B	Acceptance is existential: one accepting path is enough.
10	D	$1010 = 10$ and $10 \bmod 3 = 1$: path r_0, r_1, r_2, r_2, r_1 .
11	C	$\lambda : Q \rightarrow \Delta$, so each state has a fixed output.
12	A	$\lambda : Q \times \Sigma \rightarrow \Delta$, output is attached to transitions.
13	D	Outputs from $n + 1$ states are produced, including the start state.
14	B	Each state is split by its distinct incoming output symbols, at most $ \Delta $ copies.
15	C	$\lambda'(q, a) = \lambda(\delta(q, a))$: the output of the target state.

Quick key: 1-B, 2-C, 3-B, 4-D, 5-A, 6-C, 7-D, 8-C, 9-B, 10-D, 11-C, 12-A, 13-D, 14-B, 15-C

Practice Section B: 10 Broad Questions

Attempt each answer in about 1.5 to 2 pages in an examination style, using definitions, examples and diagrams wherever useful.

- Q1.** Define an automaton and a finite automaton. Draw the block diagram of a finite automaton and explain the function of each component.
- Q2.** What is a transition system? Explain transition diagram and transition table with an example. State and prove the property $\delta^*(q, xy) = \delta^*(\delta^*(q, x), y)$ of the extended transition function.
- Q3.** Define acceptability of a string by a finite automaton and the language accepted by it. Design a DFA over $\{0, 1\}$ that accepts all binary numbers divisible by 3, and show the acceptance of 110 and 1001 and the rejection of 101.
- Q4.** Define DFA and NFA formally. Differentiate between them in a table and give one example of each (diagram and table).
- Q5.** State and prove the theorem on the equivalence of DFA and NFA (subset construction). Explain why the DFA may need 2^n states.
- Q6.** Construct a DFA equivalent to the NFA with states $\{p, q, r\}$, start state p , final state r and transitions $\delta(p, 0) = \{p, q\}$, $\delta(p, 1) = \{p\}$, $\delta(q, 0) = \{r\}$, $\delta(q, 1) = \{r\}$, $\delta(r, 0) = \delta(r, 1) = \emptyset$. What language does it accept?
- Q7.** Define Moore machine and Mealy machine. Compare them with respect to output function, output length, and number of states. Give one example of each with the output for an input string of your choice.
- Q8.** Explain the procedure to convert a Mealy machine into a Moore machine. Convert the Mealy machine with states q_1 (start), q_2 and transitions $q_1: 0 \rightarrow q_2/1, 1 \rightarrow q_1/0$; $q_2: 0 \rightarrow q_1/1, 1 \rightarrow q_2/0$ into an equivalent Moore machine.
- Q9.** Explain the procedure to convert a Moore machine into a Mealy machine. Convert the Moore machine with states $a/0$ (start), $b/1$, $c/0$ and transitions $a: 0 \rightarrow b, 1 \rightarrow c$; $b: 0 \rightarrow c, 1 \rightarrow a$; $c: 0 \rightarrow a, 1 \rightarrow b$ into a Mealy machine, and verify both on the input 0110.
- Q10.** Describe the applications of finite automata and of Mealy and Moore machines. Design a Mealy machine for a serial binary adder and show the addition of 101 and 011.

End of Chapter 2