Chapter 1
Finite Automata and Regular Languages

LEARNING OBJECTIVES

Fundamental, Languages, Operations, Finite state machine, NFA with ∈ -moves, Conversion of NFA to DFA, Minimization of DFA, Equivalence between NFA and DFA ,Mealy and Moore machines , Equivalence of Moore and Mealy machine ,Regular languages, Constructing FA for given RE, Pumping lemma for regular sets ,Closure properties of regular sets ,Regular grammar.

FUNDAMENTALS

Alphabet: An alphabet is a finite non-empty set of symbols.

Example: Portion of a calculator: {0, 1, 2, 3 … 9, ÷, =, –, +, ×, (,)}

Note: 1. At least one symbol is necessary.

2. 'Σ' denote Alphabet.

String: A string over an alphabet 'A' is a finite ordered sequence of symbols from 'A'. The length of string is number of symbols in string, with repetitions counted.

Example: If Σ = {0 – 9, ÷, =, –,+, × (,)} then Strings valid: 12+ 34, 90 × 10, (1+ 2) × (1÷ 3)

Strings Invalid: sin (45), log (10) etc. These strings are not valid because sin ( ), log ( ) are not defined over the alphabet set.

Note: Repetitions are allowed.

Length of |12+ 34| = 5(1, 2,+ , 3, 4)

- The Empty string denoted by 'Ɛ ', is the (unique) string of length zero.

Note: Empty string, Ɛ ≠ empty set, Ø .

- If S and T are sets of strings, then ST = {xy|x∈ S and y ∈T} Given an alphabet A,

$A^o = \{Ɛ\}$

$A^{n+1} = A.A^n$

...

$A^* = \bigcup_{n=0}^{\infty} A^n$

LANGUAGES

- A language 'L' over **Σ** is any finite or infinite set of strings over **Σ**.
- The elements in L are strings – finite sequences of symbols.
- A language which does not contain any elements is called 'empty language'.

Note: Empty language, { } ≠{Ɛ }, empty string because { } =Ø≠ Ɛ i.e., Empty language resembles empty set i.e.,Ø .

- A language L over an alphabet A is subset of A^* i.e., L⊂ A^*.

Example 1: Language (L) for strings that consists of only 0's or only 1's and have an odd length over alphabet {0, 1} is

(A) {0, 1, 00, 11, 000, 111 ,...}

(B) {00, 11, 01, 10 ,…}

(C) {000, 101, 110, 111 ,…}

(D) {0, 1, 000, 111, 11111, 00000 ,…}

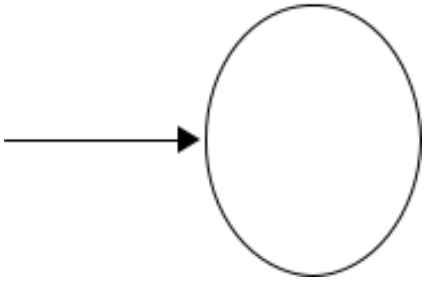
Solution: (D)

Only 0's → should have only 0's. It should not be combination of 0's and 1's.

Only 1's → should have only 1's. It should not be combination of 0's and 1's.

Odd length → only odd number of 0's or odd number of 1's i.e., length of string should be odd.

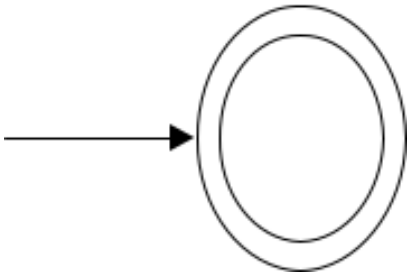
An Empty Languages :An empty language is a language which does not accept any strings includinge. The Finite automata for empty language can be represented as



(i.e., One state, non-accepting and no transitions).

A language which only accepts (Ɛ)

E: The language which only accepts ‘Ɛ ’ can be represented as



This machine accepts E – only.

Σ^*: The set of all strings over an alphabet will be denoted by Σ^*.

Σ^+: This will denote the set $\Sigma^* - \{ Ɛ \}$.

Ex: If Σ = {0, 1} then

Σ^* = { Ɛ , 0, 1, 00, 01, 10, 11, 000, 001,… }

Σ^+= {0, 1, 00, 01, 10, 11, 000, 001,}

OPERATIONS

Operations on strings

1.Concatenation: Combines two strings by putting one after other.

Example 2: Two strings are defined as x = java, y = script.

The concatenation (x.y) of two strings results in _____.

(A) scriptjava (B) javascript

(C) jascriptva (D) scrijavapt

Solution: (B)

x.y = java.script = javascript

Note: Concatenation of empty string with any other string gives string itself.

i.e., x. Ɛ = Ɛ .x = x

2.Substring: If 'w' is a string, then 'v' is a substring of 'w' if there exists string x and y such that w = xvy.

'x' is called 'prefix' and y is called suffix of w.

Example 3: String, w = 'gymnastics' is defined with prefix, x = 'gym' and suffix, y = 'cs'. The substring of the given string is _____

(A) nasti (B) mnas

(C) gymnastics (D) ics

Solution: (A)

Because, w = xvy

⇒ gymnastics = gymvcs

∴ v = nasti

3.Kleen star operation: Let 'w' be a string, w^* is set of strings obtained by applying any number of concatenations of w with itself, including empty string.

Example: a^* = { , a, aa, aaa,…}

4.Reversal: If 'w' is a string, then w^R is reversal of string spelled backwards.

Rules:

$x = (x^R)^R$

$(xz)^R = z^R.x^R$

Example 4: A string, x is defined as, x = butter. Then $(x^R)^R$ is _____

(A) butter (B) rettub

(C) butret (D) retbut

Solution: (A)

x → butter

$x^R \rightarrow$ rettub

$(x^R)^R \rightarrow$ butter.

Operations on languages

1.Union: Given some alphabet Σ, for any two languages, L_1, L_2 over Σ, the union $L_1 \cup L_2$ of L_1 and L_2 is the language,

$L_1 \cup L_2 = \{w \in \Sigma^* | w \in L_1 \text{ or } w \in L_2\}$

2.Intersection: Given some alphabet Σ , for any two languages L_1,L_2 over Σ the intersection $L_1 \cap L_2$ of L_1 and L_2 is language,

$L_1 \cap L_1 = \{w \in \Sigma^* | w \in L_1 \text{ and } w \in L_2\}$

3.Difference: Given some alphabet Σ , for any two languages L_1,L_2 over Σ the intersection $L_1 - L_2$ of L_1 and L_2 is language,

$L_1 - L_1 = \{w \in \Sigma^* | w \in L_1 \text{ and } w \notin L_2\}$

Note: Difference is also called 'Relative Complement.'

A special case of difference is obtained when L1 = Σ *, in which case. Complement $\overline{L}$ of language, L is defined as, $\overline{L} = \{w \in \Sigma^* | w \notin L\}$

4.Concatenation: Given an alphabet Σ , for any two languages L_1, L_2 over Σ, the concatenation L1 L2 of L1 and L2 is language

$L_1L_2 = \{w \in \Sigma^* | \exists u \in L_1, \exists v \in L_2, w = uv\}$

Properties:

$L\emptyset = \emptyset = \emptyset L$

$L\ \{\varepsilon\} = L = \{\varepsilon\}\ L$

$(L_1 \cup \{\varepsilon\})L_2 = L_1L_2 \cup L_2$

$L_1 (L_2 \cup \{\varepsilon\}) = L_1L_2 \cup L_1$

$L^n L = LL^n = L^{n+1}$

Note: $L_1L_2 \neq L_2L_1$

Example 5: Let $L_1 = \{00, 11\}$, $L_2 = \{01, 10\}$. Then $L_1 o\ L_2$ _____

A. {00, 11, 01, 10}
B. {0001, 0010, 1101, 1110}
C. {0001, 0010, 11, 01, 10}

D. {00, 1101, 1110, 11, 10}

Solution: (B)

$L_1 o L_2$ = {00, 11} o {01, 10} = {00.01, 00.10, 11.01, 11.10} = {0001, 0010, 1101, 1110}

5.Kleen * closure (L^*): Given an alphabet Σ, for any language L over Σ, the * closure L^* of L is language, $L^* = U_{n\geq 0} L^n$

6.Kleen+ closure (L^+): The kleen +closure, L^+ of L is the language, $L^+ = U_{n\geq 1} L^n$

$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots L^n \cup \ldots$

$L^+ = L^1 \cup L^2 \cup L^3 \ldots \cup L^n \cup \ldots$

Properties:

$\emptyset^* = \{\varepsilon\}$

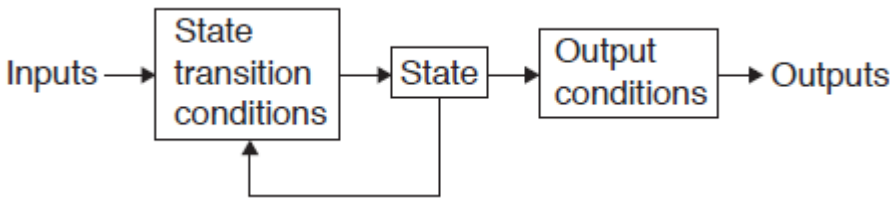
$L^+ = L^* L$

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

$L^* L^* = L^*$

FINITE STATE MACHINE (FSM)

- FSM is simplest computational model of limited memory computers.
- FSM is designed to solve decision problems i.e., to decide whether given input satisfies certain conditions.
- The next state and output of a FSM is a function of input and of current state.

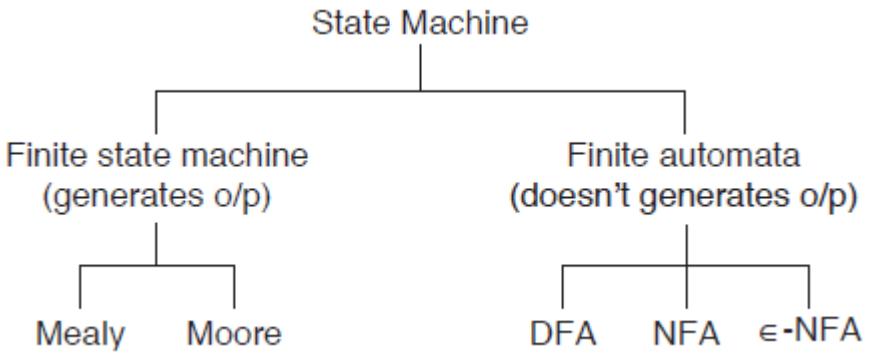


Types of FSM:

1. Melay machine.
2. Moore machine

Finite Automata (FA):

- FA is a state machine that comprehensively captures all possible states and transitions that a machine can take while responding to a stream (sequence) of input symbols.
- FA is recognizer of 'regular languages'.



Types of FA

1. **Deterministic Finite Automata (DFA):**

- DFA machine can exists in only one state at any given time.
- DFA is defined by 5-tuple: {Q, Σ, q0, F, δ}, where

Q → Finite number of states (elements)

Σ → Finite set of symbols (alphabets)

q_o → Start/Initial state

F → Set of final states.

δ → Transition function, which is a mapping between

δ: Q × Σ → Q.

How to use DFA:

Input: A word w in Σ^*

Question: Is w acceptable by DFA?

Steps:

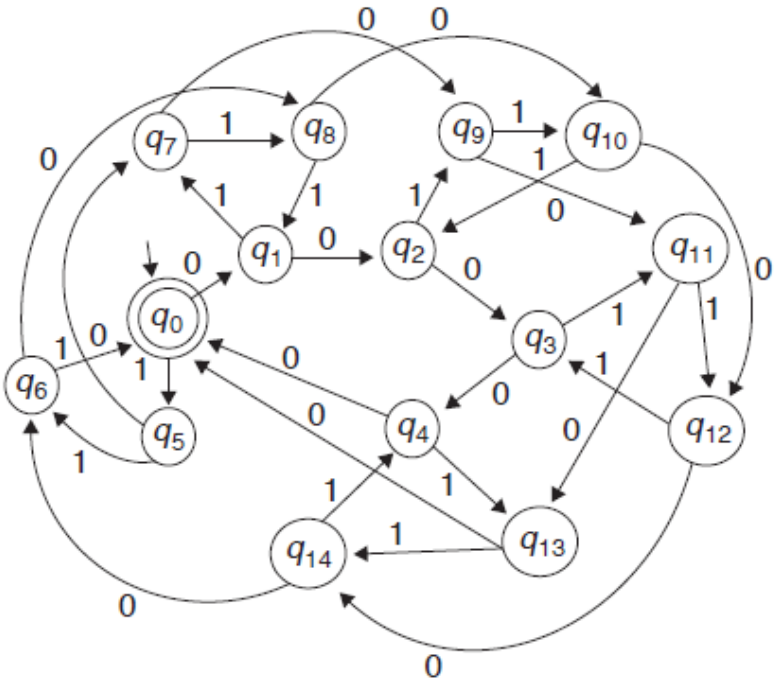
- Start at 'initial state', q_o.
- For every input symbol in sequence w, do.
- Compute the next state from current state, given the current input symbol in w and transition function.
- If after all symbols in 'w' are consumed, the current state is one of the final states (f) then accept 'w';
- Otherwise, reject w.

Transition diagram: State machines are represented by directed graphs called transition (state) diagrams.

- The vertices denoted by single circle represent the state and arcs labeled with input symbol correspond to transition.
- The final states are represented with double circles.

Transition Table: Transition function can be represented by tables.

Example 6: The following finite state machine accepts all those binary strings in which the numbers of 0's and 1's are respectively.



(A) Divisible by 3 and 2 (B) Odd and even

(C) Divisible by 5 and 3 (D) Divisible by 2 and 3

Solution: (C)

Number of 0's is divisible by 5.

Number of 1's is divisible by 3.

Transition Table

Current State	0	1
$\rightarrow (q_0)$	q_1	q_5
q_1	q_2	q_7
q_2	q_3	q_9
q_3	q_4	q_{11}
q_4	q_0	q_{13}
q_5	q_7	q_6
q_6	q_8	q_0
q_7	q_9	q_8
q_8	q_{10}	q_1
q_9	q_{11}	q_{10}
q_{10}	q_{12}	q_2
q_{11}	q_{13}	q_{12}
q_{12}	q_{14}	q_3
q_{13}	q_0	q_{14}
q_{14}	q_6	q_4

Note: Minimum number of states for k-divisibility is k-states.

In above example, $q_0 - q_{14} \rightarrow 15$ – states.

$\therefore 5 \times 3 = 15$

The given binary strings have number of 0's divisible by 5 and number of 1's divisible by 3.

2. Non-deterministic finite Automata (NFA):

• The machine can exist in multiple states at the same time.

• Each transition function maps to a set of states.

• NFA is defined by 5-tuple: $\{Q, \Sigma, q_o, F, \ \}$, where

Q → Finite number of states (elements)

$\Sigma \rightarrow$ Finite set of symbols. (Alphabets)
$q_o \rightarrow$ Start/Initial state
$F \rightarrow$ Set of final states.
$\delta \rightarrow$ Transition function which is a mapping between
$\delta = Q \times \Sigma \rightarrow 2^Q$

How to use NFA:

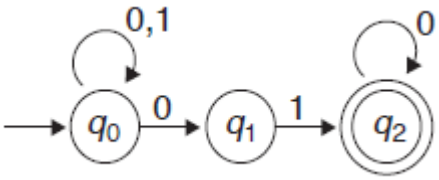
Input: a word w in Σ^*

Question: Is w accepted by NFA?

Steps:

- Start at 'start state' q_o.
- For every input symbol in the sequence, w does.
- Determine all possible next states from current state, given the current input symbol in w and transition function.
- If after all symbols in w are consumed, at least one of the current states is a final state then accept w.
- Otherwise, reject w.

Example 7: What is the language, L generated by the below NFA, given strings defined over alphabet, $\Sigma = \{0, 1\}$.



(A)Strings that end with '0'

(B) Strings that start with '0' and end with '0'

(C) Strings that contain '01' as substring

(D) Strings that contain '01' as substring and end with '0'

Solution: (D)

State	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\varnothing$	q_2
q_2	$\{q_2\}$	$\varnothing$

String: 0100100

$$q_0 \xrightarrow{0} \{q_0, q_1\}$$

$$q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{0} \{q_0, q_1\}$$

$$q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{0} \{q_0, q_1\}$$

$$\xrightarrow{q_0 1} q_0 \xrightarrow{0} q_0 \xrightarrow{0} \{q_0, q_1\} \text{ (Non-deterministic)}$$

$$q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{0} \{q_0, q_1\}$$

$$q_1 \xrightarrow{1} q_2 \xrightarrow{0} q_2 \xrightarrow{0} q_2$$

Difference between NFA and DFA

DFA	NFA
1. All transitions are deterministic i.e., each transition leads to exactly one state.	1. Transitions could be non-deterministic i.e., a transition could lead to a subset of states.
2. For each state, the transition on all possible symbols should be defined.	2. For each state, not all symbols necessarily have to be defined.
3. Accepts input if last state is in 'F'.	3. Accepts input if one of last states is in 'F'.
4. Practical implementation is feasible.	4. Practical implementation has to be deterministic (so needs conversion to DFA).

Relation between DFA and NFA

• A language 'L' is accepted by a DFA if and only if it is accepted by a NFA.

• Every DFA is special case of a NFA.

Example 8: Let N_f and D_f denote the classes of languages accepted by non-deterministic finite automata and deterministic finite automata respectively. Which one of following is true?

(A) $D_f \subset N_f$ (B) $D_f \supset N_f$

(C) $D_f = N_f$ (D) $D_f \in N_f$

Solution: (C)

According to 'subset construction', every language accepted by NFA is also accepted by some DFA.

$\therefore D_f = N_f$

NFA WITH ∈ -MOVES

• ∈-transitions in finite automata allows a state to jump to another state without consuming any input symbol.

Conversion and Equivalence:

∈-NFA →NFA → DFA

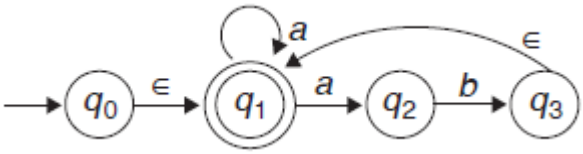
NFA without ∈-moves:

• Two FA, $N_\in$ and N are said to be equivalent, if $L(N_\in) = L(N)$ i.e., any language described by some $N_\in$, there is an N that accepts the same language.

• For $N_\in = (Q, Z, \delta, q_0, F)$ and $N = (Q, \Sigma', \delta', q_0, F')$, Find
• $\delta'(q, a) =$ ∈-closure (δ (∈-closure(q), a))
• $F' = \{F \cup \{q_0\}\}$, if ∈-closure (q_0) contains a member of F = F, otherwise.

Note: When transforming $N_\in$ to N, only transitions are required to be changed and states remains same.

Example 9: Consider following NFA with ∈-moves.



If given NFA is converted to NFA without ∈-moves, which of following denotes set of final states?

(A) $\{q_0, q_1\}$ (B) $\{q_1, q_2\}$

(C) $\{q_1, q_2, q_3\}$ (D) $\{q_1\}$

Solution: Let $N = (Q, \Sigma^1, \delta^1, q_0, F^1)$

$F^1 = F \{q_0\}$

Ɛ -closure $(q_0) = \{q_0, q_1\}$

$\therefore F^1 = \{q_1\} \{q_0, q_1\} = \{q_0, q_1\}$

Conversion $N_{\in} \rightarrow N$:

To compute, δ^1

∈-closure $(q_0) = \{q_0, q_1\}$, ∈-closure $(q_3) = \{q_3, q_1\}$

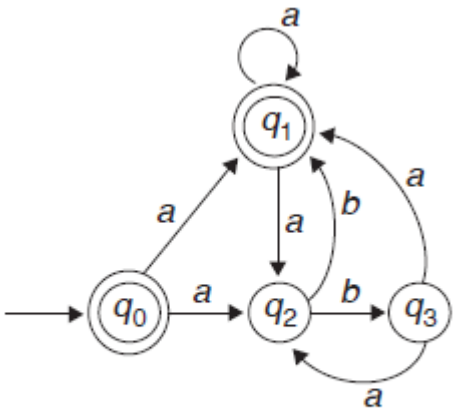
$\delta^1(q_0, a) = \{q_1, q_2\}$, $\delta^1(q_0, b) = Ø$, $\delta^1(q_2, a) = Ø$.

$\delta^1(q_1, a) = \{q_1, q_2\}$, $\delta^1(q_1, b) = Ø$, $\delta^1(q_2, b) = \{q_1, q_3\}$

$\delta^1(q_3, a) = \{q_1, q_2\}$, $\delta^1(q_3, b) = Ø$

Transition Table

State \ Input	*a*	*b*
$\rightarrow (q_0)$	$\{q_1, q_2\}$	∅
(q_1)	$\{q_1, q_2\}$	∅
q_2	∅	$\{q_1, q_3\}$
q_3	$\{q_1, q_2\}$	∅



Transition diagram

CONVERSION OF NFA TO DFA

Let a NFA be defined as, $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$

The equivalent DFA, $D = (Q_D, \Sigma, \delta_D, q_0, F_D)$ where:

Step I: $Q_D = 2^{QN}$; i.e., Q_D is set of all subsets of Q_N i.e., it is power set of Q_N.

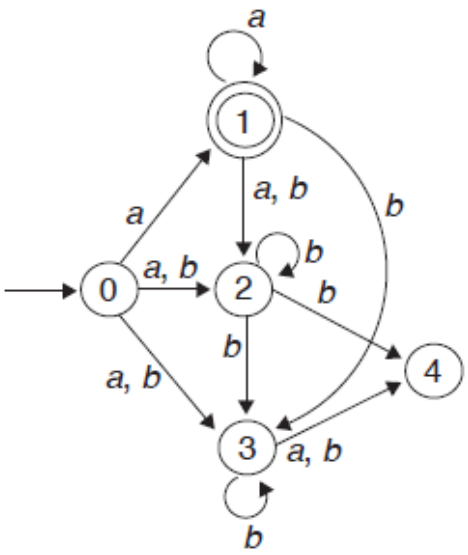
Step II: F_D is set of subsets S of Q_N such that $S \cap F_N \neq \emptyset$. i.e., F_D is all sets of N's states that include atleast one accepting state of N.

Step III: For each set, $S \leq Q_N$ and for each input symbol a in Σ : $\delta_D(S, a) = \cup_{P \in S} \delta_N(P, a)$

That is, to compute $\delta_D(S, a)$, look at all states P in S, see what states N goes to starting from P on input a, and take the union of all those states.

Note: For any NFA, N with 'n' states, the corresponding DFA, D can have 2n states.

Example 10: What is the number of final states in DFA constructed from the given NFA?



(A) 1 (B) 2

(C) 3 (D) 4

Solution:

Transition Table of NFA

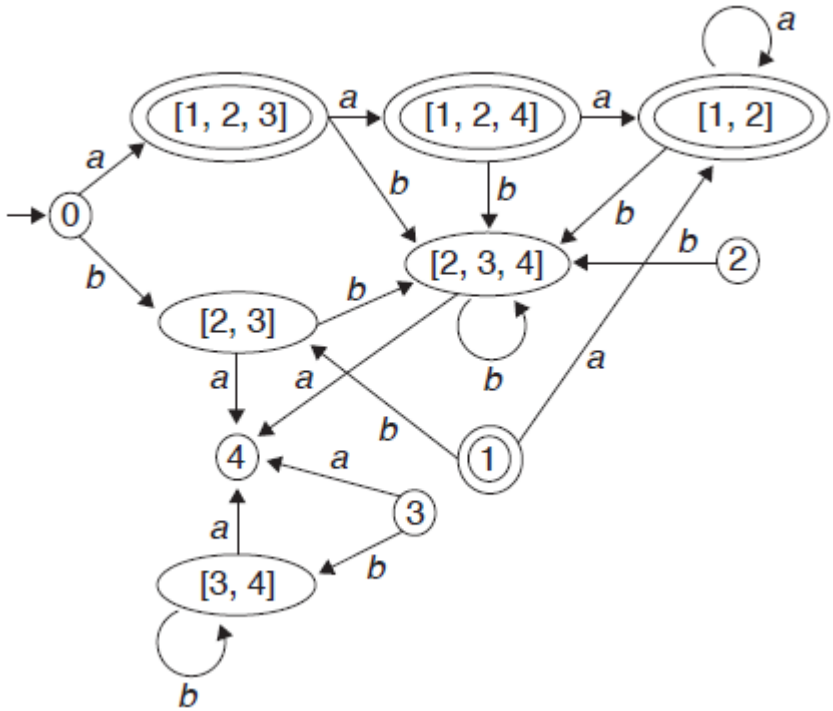
State \ Input	a	b
→ 0	{1, 2, 3}	{2, 3}
(1)	{1, 2}	{2, 3}
2	∅	{2, 3, 4}
3	{4}	{3, 4}
4	∅	∅

Transition Table of DFA

State \ Input	a	b
→0	[1, 2, 3]	[2, 3]
(1)	[1, 2]	[2, 3]
2	∅	[2, 3, 4]
3	4	[3, 4]
4	∅	∅
([1, 2])	[1, 2]	[2, 3, 4]
[2, 3]	[4]	[2, 3, 4]
[3, 4]	[4]	[3, 4]
([1, 2, 3])	[1, 2, 4]	[2, 3, 4]
([1, 2, 4])	[1, 2]	[2, 3, 4]
[2, 3, 4]	[4]	[2, 3, 4]

Hence final states in obtained DFA is '4'.

DFA is: Choice (D)

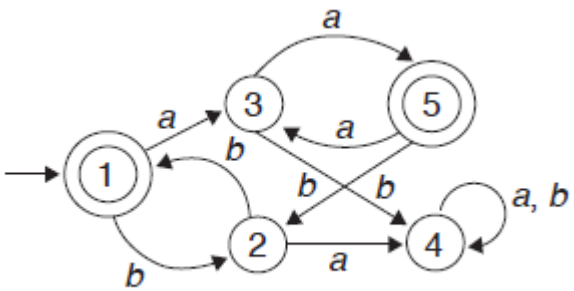


MINIMIZATION OF DFA

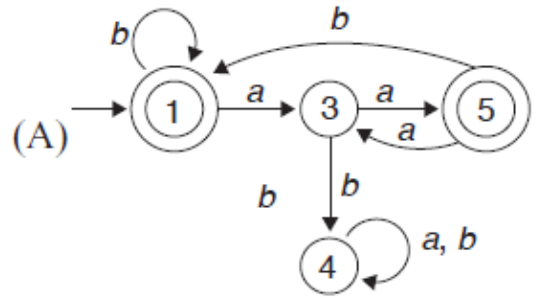
Given a DFA, $M = (Q, \Sigma, \delta, q_0, F)$ we construct a reduced DFA, $M' = (Q', \Sigma', \delta', q'_0, F')$ 0 as follows

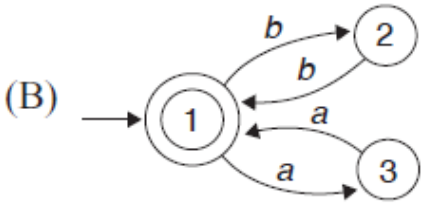
1. Remove all inaccessible states. All states that are unreachable from the initial state are removed.
2. Consider all pairs of states (p, q), If $p \in F$ and $q \in F$ or vice versa mark the pair (p, q) as distinguishable.
3. Repeat until no previously unmarked pairs are marked. For all pairs (p, q) and all $a \in \Sigma$, compute $\delta(p, a) = p_a$ and $\delta(p, q) = q_a$. If the pair (p_a, q_a) is marked as distinguishable mark (p, q) as distinguishable.
4. Find the sets of all indistinguishable states, say $\{q_i, q_j, \ldots q_k\}$, $\{q_\ell, q_m, \ldots q_n\}$, etc. For each set $\{q_i, q_j, \ldots q_k\}$ of such indistinguishable states, create a state labelled ij …k for M.
5. For each transition rule of M of the from $\delta(q_r, q) = q_p$, find the sets to which q_r and q_p belong. If $qr \in \{q_i, q_j, \ldots q_k\}$ and $q_p \in \{q_\ell, q_m, q_n\}$, add a rule to δ: $\delta'(ij\ldots k, a) = \ell m\ldots n$.

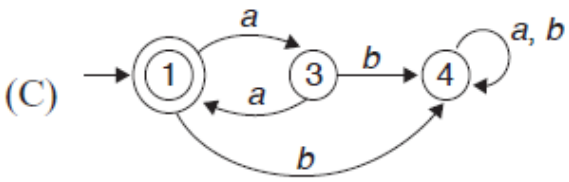
Example 11: A DFA with alphabet $\Sigma = \{a, b\}$ is given below:

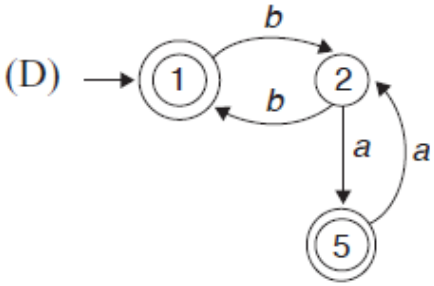


Which of the following is valid minimal DFA which accepts same language as given DFA?









Solution: (B)

Initially, {1, 5}, {2, 3, 4}

Depending on next states and inputs, the partitions of states can be as: {{1, 5}, {2}, {3}, and {4}}

Since, 1 to 5 have same transition, unite {1, 5}

State 4 is dead state → It has transition only to itself.

Since, {2}, {3} are singletons, they exist.

∴ States in minimized DFA are {1, 2, and 3}

{1} → {1, 5}

For transitions, since $1 \xrightarrow{a} 3$, $1 \xrightarrow{b} 2$ in given DFA, in minimized DFA, transitions are added from $1 \xrightarrow{a} 3$, $1 \xrightarrow{b} 2$. Also, since $2 \xrightarrow{b} 1$, $3 \xrightarrow{a} 1$ in given DFA, the minimized DFA, transitions are added from $2 \xrightarrow{b} 1$, $3 \xrightarrow{a} 1$.

EQUIVALENCE BETWEEN NFA AND DFA

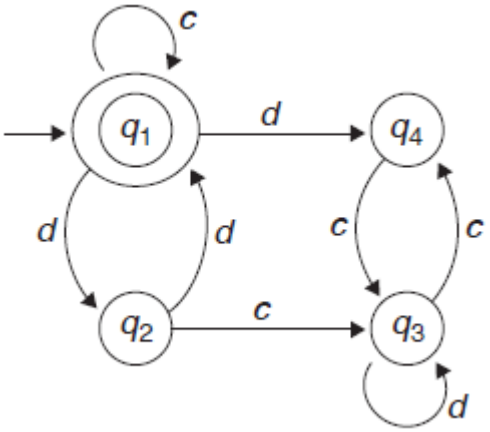
There is a DFA_D for any NFA_N i.e.,

L (D) = L (N).

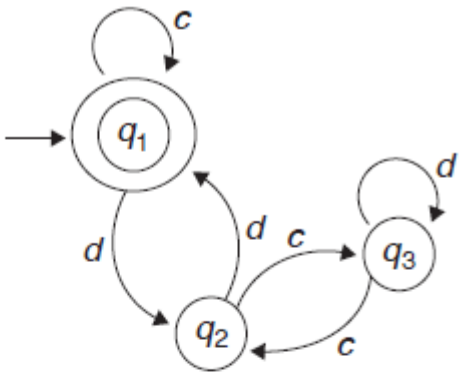
Construction:

- In DFA or NFA, whenever an arrow is followed, there isa set of possible states. This set of states is a subset of Q.
- Track the information about subsets of states that can be reached from initial state after following arrows.
- Consider each subset of states of NFA as a state of DFAand every subset of states containing a final state as a finalstate of DFA.

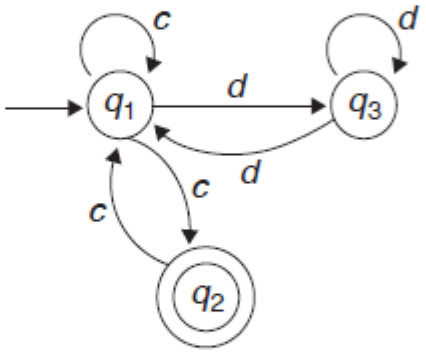
Example 12: Which of following is equivalent DFA for the NFA given below:



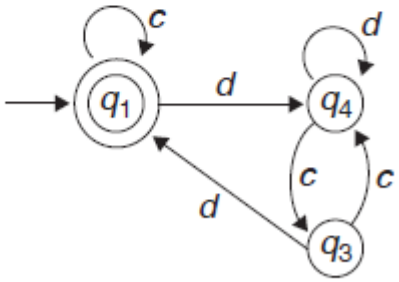
(A)



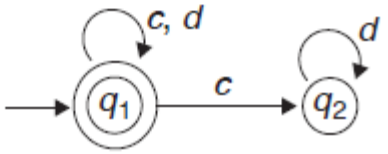
(B)



(C)



(D)



Solution: (A)

Transition Table of NFA

δ	**c**	**d**
$\rightarrow(q_1)$	q_1	$\{q_2, q_4\}$
q_2	q_3	q_1
q_3	q_4	q_3
q_4	q_3	$\varnothing$

Transition Table of DFA

δ	**c**	**d**
$\rightarrow(q_1)$	q_1	q_2
q_2	q_3	q_1
q_3	q_2	q_1

Common Table

δ	**c**	**d**
(q_1, q_1)	(q_1, q_1)	(q_2, q_4, q_2)
(q_2, q_2)	(q_3, q_3)	(q_1, q_1)
(q_3, q_3)	(q_4, q_2)	(q_3, q_3)
(q_4)	q_3	$\varnothing$

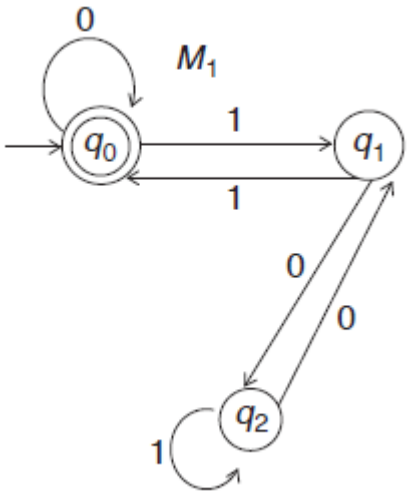
Equivalence of Finite Automatas:

- Two automatas A and B are said to be equivalent if both accept exactly the same set of input strings.
- If two automatas M_1 and M_2 are equivalent then

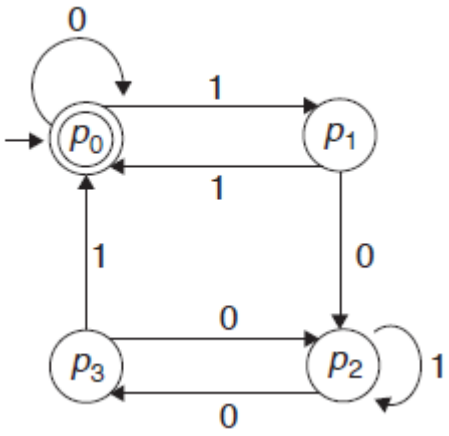
(i) If there is a path from the start state of M_1 to a final state of M_1 labeled $a_1a_2 \ldots a_k$ then there is a path from the start state of M_2 to the final state of M_2 labeled $a_1a_2 \ldots a_k$.

(ii) If there is a path from the start state of M_2 to a final state M_2 labeled $b_1b_2 \ldots b_i$ then there is a path from the start state of M_1 to the final state of M_1 labeled $b_1b_2 \ldots b_i$.

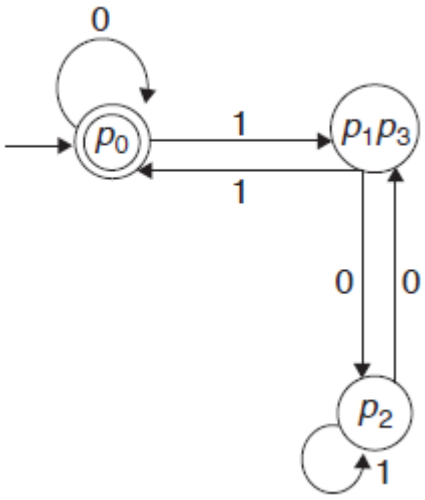
Example:



M2:



In M_2, states p_1 and p_3 are equivalent (as both are reaching either final or non-final states with same input). After minimizing M_2, we will get



∴ M_1 and M_2 are equivalent.

Union: The union of two languages L and M is the set of

strings that are in both L and M.

Ex: L = {0, 1}, M = {111}

L U M = {0, 1,111}.

Concatenation: The concatenation of Languages L and M is the set of strings that can be formed by taking any string in L and concatenating it with any string in M.

Example: L = {0, 1}, M = { Ɛ, 010}

LM = {0, 1, 0010, 1010}.

Closure, Star or Kleen star of a language L:

Kleen star is denoted as L^*. It represents the set of strings that can be formed by taking any number of strings from L with repetition and concatenating them. It is a Unary operator.

L^0 is the set; we can make selecting zero strings from L.

L^0 ={ ε }

L^1 is the language consisting of selecting one string from L.

L^2 is the language consisting of concatenations selecting two strings from L.

…

L^* is the union of L^0, L^1 , …L^∞.

Ex: L ={0,10}

L^* ={0,00,000,10,010, …}

Intersection:

Let two DFAs M_1 and M_2 accept the languages L_1 and L_2.

M_1 =(Q_1, Σ, δ_1, q_0^1, F1)

M2=(Q_2, Σ, δ_2, q_0^2, F2)

The intersection of M_1 and M_2 can be given as

M =(Q1, Σ, δ, q_0 F)

Q =Pairs of states, one from M_1 and one from M_2 i.e.,

Q ={(q_1, q_2) | $q_1 \in Q_1$ and $q_2 \in Q_2$}

Q =$Q_1 \times Q_2$.

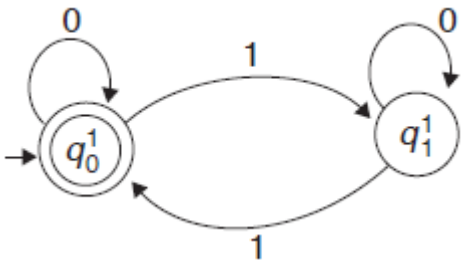
q_0=(q_0^1, q_0^2)

δ (q_i^1 q_j^2), x) =(δ_1 (q_i^1, x), δ_2 (q_j^2, x))

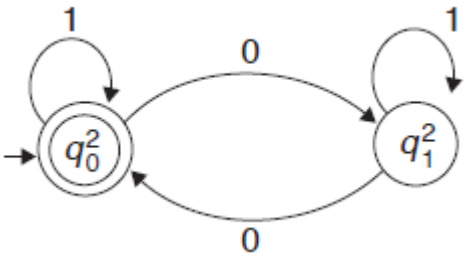
F ={(q_1, q_2) | $q_1 \in F_1$ and $q_2 \in F_2$}

Example:

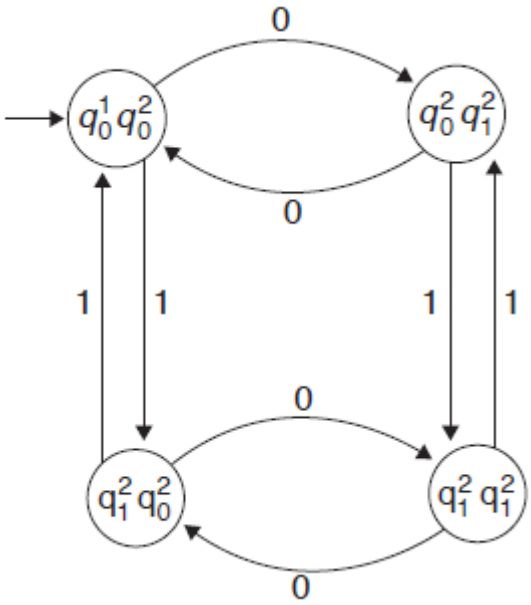
M_1: Strings with even number of 1's.



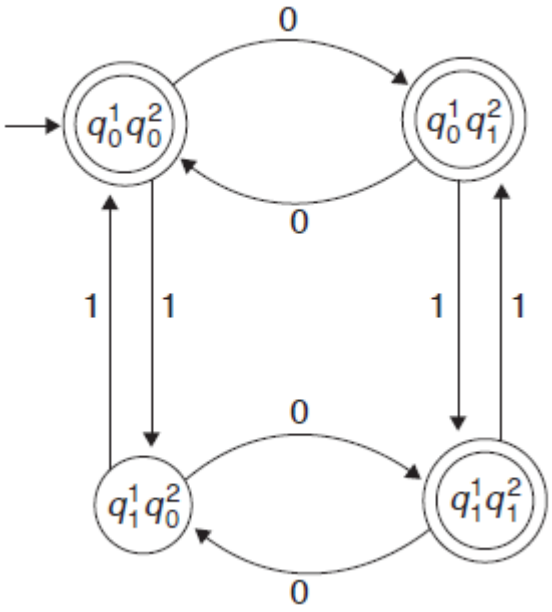
M_2: Strings with odd number of 0's.



M1 ∩ M2: Strings with even number of 1's and odd number of 0's.



Union of M1 and M2:



Difference: The difference of L_1 and L_2 can be given as

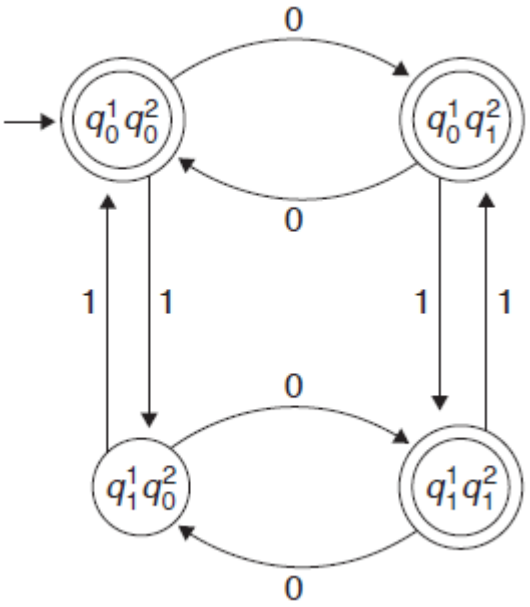
$L_1 - L_2$ with M = (Q, Σ, δ, q_0, F).

$Q = Q_1 \times Q_2$

q0= (q_0^1, q_0^2)

$\delta((q_i^1, q_j^2), x) = (\delta_1(q_i^1, x), \delta_2(q_j^2, x))$

$F = \{(q_1, q_2) \mid q_1 \in F_1 \text{ and } q_2 \notin F_2\}$



Reversing a DFA:

• M is a DFA which recognizes the language L.

• M^R will accept the language L^R.

To construct M^R:

• Reverse all transitions

• Turn the start state to final state

• Turn the final states to start state.

• Merge states and modify the FA,

such that the resultant contain a single start state.

MEALY AND MOORE MACHINES

Moore Machine

A moore machine is a finite state machine, where outputs are determined by current state alone.

A Moore machine associates an output symbol with each state and each time a state is entered, an output is obtained simultaneously. So, first output always occurs as soon as machine starts.

Moore machine is defined by 6-tuples:

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

$Q \rightarrow$ Finite set of states

$\Sigma \rightarrow$ Finite set of input symbols

$\Delta \rightarrow$ It is an output alphabet

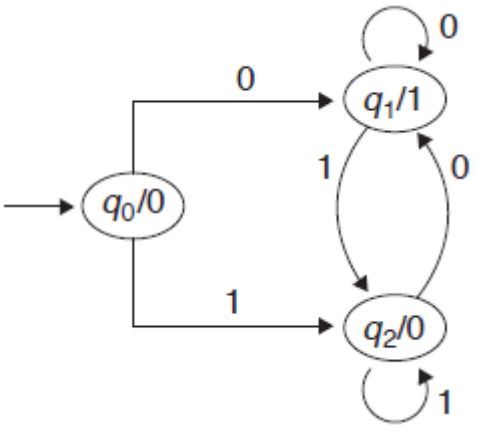
$\delta \rightarrow$ Transition function, $Q \times \Sigma \rightarrow Q$ (state function)

$\lambda \rightarrow$ Output function, $Q \rightarrow \Delta$ (machine function)

$q_0 \rightarrow$ Initial state of machine

Note: The output symbol at a given time depends only on present state of moore machine.

Example 13: The language generated by the following moore machine is:



(A) 2's complement of binary number.

(B) 1's complement of binary number.

(C) Has a substring 101.

(D) Has a substring 110.

Solution: (B)

Binary number: 1011

1's complement: 0100

$$q_0 \xrightarrow{1/0} q_2, \xrightarrow{0/1} q_1 \xrightarrow{1/0} q_2 \xrightarrow{1/0} q_2$$

$$1 \rightarrow 0, 0 \rightarrow 1, 1 \rightarrow 0, 1 \rightarrow 0$$

Mealy Machine

- A mealy machine is a FSM, where outputs are determined by current state and input.
- It associates an output symbol with each transition and the output depends on current input.
- Mealy machine is defined on 6-tuples: $(Q, \Sigma, \delta, q_0, \Delta, \lambda)$, where

Q – Finite set of states.

Σ – Finite set of input symbols.

δ – $(Q \times \Sigma \rightarrow Q)$ is transition function.

$q_0 \rightarrow q_0 \in Q$ is initial state.

$\Delta \rightarrow$ Finite set of output symbols.

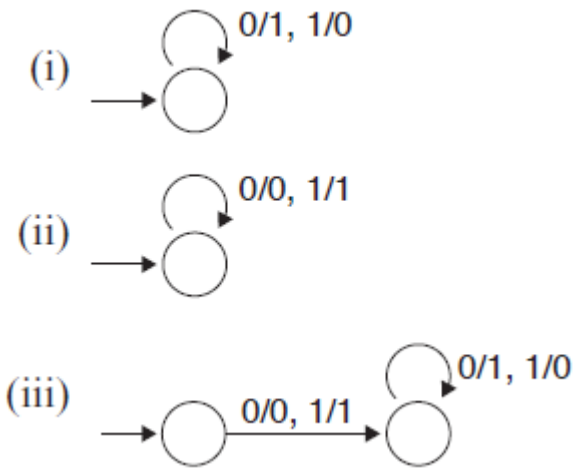
$\lambda \rightarrow$ Output function, λ $(Q \rightarrow \Delta)$

Note: In Moore machine, for input string of length n, the output sequence consists of (n + 1) symbols.

In Mealy machine, for input string of length n, the output sequence also consists of 'n' symbols.

Example 14: Let $(Me)^2$ mean that given a Mealy machine, an input string is processed and then output string is immediately fed into the machine (as input) and reprocessed.

Only this second resultant output is considered as the final output of $(Me)^2$. If final output string is same as original input string then $(Me)^2$ has an identity property. Consider following machines



Which of above machines have identity property?

(A) (i) only

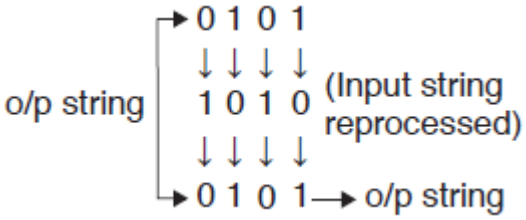
(B) (i) and (ii) but not (iii)

(C) (i) and (iii) but not (ii)

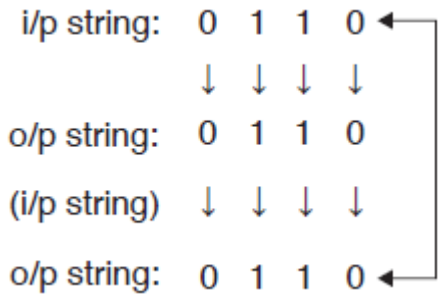
(D) All have identity property

Solution: (D)

(i) Consider i/p string



(ii)



(iii)

EQUIVALENCE OF MOORE AND MEALY MACHINE

(a) Mealy machine equivalent to Moore machine:

If $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ is a Moore machine, then there is a Mealy machine M_2 equivalent to M_1.

Proof: Let $M_2 = (Q, \Sigma, \Delta, \delta, \lambda^1, q_0)$ and define λ^1 (q, a) to be λ (δ (q, a)) for all states q and input symbol 'a'.

Then M_1 and M_2 enter the same sequence of states on the same input, and with each transition M_2 emits the o/p that M_1 associates with the state entered.

Let us consider Mealy Machine

	Next State			
Present State	Input State	$a = 0$ Output	Input State	$a = 1$ Output
$\rightarrow q_1$	q_3	0	q_2	0
q_2	q_1	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0

To convert the Mealy machine to Moore machine,

- We look into the next state column for any state, say q_i and determine the number of different outputs associated with q_i in next column.
- Split q_i into several different states, the number of such states being equal to the number of different outputs associated with q_i.

Present State	Next State			
	Input State	$a = 0$ Output	Input State	$a = 1$ Output
$\rightarrow q_1$	q_3	0	q_{20}	0
q_{20}	q_1	1	q_{40}	0
q_{21}	q_1	1	q_{40}	0
q_3	q_{21}	1	q_1	1
q_{40}	q_{41}	1	q_3	0
q_{41}	q_{41}	1	q_3	0

The pair of states and outputs in the next state column can be rearranged as:

Present state	Next State		output
	$a = 0$	$a = 1$	
$\rightarrow q_1$	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	1

Moore machine equivalent to Mealy machine

Let $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ be a Mealy machine. Then there is a machine M_2 equivalent to M_1

Proof: Let $M_2 = (Q \times \Delta, \Sigma, \Delta, \delta^1, \lambda^1, [q_0, b_0])$, where b_0 is an arbitrary λ selected member of Δ.

That is, the states of M_2 are pairs [q, b] consisting of a state of M_1 and output symbol, Define δ^1 ([q, b], a) = [(q, a), λ,(q, a)] and λ^1([q, b]) =b.

The second component of a state [q, b] of M_2 is the output made by M_1 on some transition into state q.

Only the first components of M_2's states determine the moves made by M_2.

Every induction on 'n' shows that if M_1 enters states q_0, q_1 … q_n on inputs a_1, a_2 …a_n and emits output b_1, b_2, …b_n then M_2 enters states $[q_0, b_0]$, $[q_1, b_1]$ … $[q_n, b_n]$ and emits outputs b_0, b_1 … b_n.

Let us consider the Moore machine

Present State	Next State		Output
	$a = 0$	$a = 1$	
$\rightarrow q_0$	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

- To convert Moore into Mealy machine, we must follow the reverse procedure of converting Mealy machine into Moore machine.
- For every input symbol we form, the pair consisting of the next state and the corresponding output and reconstruct the table for Mealy machine.
- For example, the state q_3 and q_1 in the next state column should be associated with outputs 0 and 1, respectively.

The Transition table for Mealy machine is:

Present state	Next State			
	$a = 0$ state output		$a = 1$ state output	
$\rightarrow q_0$	q_3	0	q_1	1
q_1	q_1	1	q_2	0
q_2	q_2	0	q_3	0
q_3	q_3	0	q_0	0

REGULAR LANGUAGES

The set of regular languages over an alphabet Σ is defined recursively as below. Any language belonging to this set is a regular language over Σ.

Definition of set of regular languages

- Basis clause: Ø, { Ɛ}, {a} for any symbol a ∈ Σ, are regular languages.
- Inductive clause:If L_r and L_s are regular languages, then $L_r \cup L_s$, $L_r . L_s$, L_r^* are regular languages.
- External clause: Nothing is a regular language, unless it is obtained from above two clauses.

Regular language: Any language represented by regular expression(s) is called a regular language.

Ex: The regular expression a^* denotes a language which has { Ɛ, a, aa, aaa, …}

Regular expression

- Regular expressions are used to denote regular languages.
- The set of regular expressions over an alphabet Σ is defined recursively as below. Any element of that set is a regular expression.
- Basis clause: Ø,∈, a are regular expression corresponding to languages Ø, {∈}, {a} respectively where a is an element of Σ.
- Inductive clause: If r and s are regular expression corresponding to languages Lr and L_s then (r + s), (rs) and (r^*) are regular expressions corresponding to the languages $L_r \cup L_s$, $L_r . L_s$ and L_r^* respectively.
- External clause: Nothing is a regular expression, unless it is obtained from above two clauses.

Closure property of regular expressions

The iteration or closure of a regular expression R, written as R^* is also a regular expression.

Ex: Σ = {a} then a^* denotes the closure of Σ.

$a^* = \{ Ɛ, a, aa, aaa, \ldots\}$

Conventions on regular expressions

1. The operation '*' has highest precedence over concatenation, which has precedence over union (+).
   i.e., RE $(a + (b(c^*))) = a + bc^*$
2. The concatenation of K r's, where r is a regular expression is written as r^k. The language corresponding to r^k is L_r^k. Where L_r is language corresponding to regular expression r i.e., $rr = r^2$
3. r^+ is a regular expression to represent L_r^+

Note: A regular expression is not unique for a language i.e., regular language corresponds to more than one regular expression.

Example 15: Give regular expression for set of strings which either have 'a' followed by some b's or all b's also containing 'Ɛ'.

(A) $b^* + ab^*$ (B) $a^* + ba^*$

(C) $(Ɛ) + (Ɛ + a) b^+$ (D) $b^* + ab^* + Ɛ$

Solution: (C)

The regular expression is, $r = ab^+ + b^+ + Ɛ = b^+ (a + Ɛ) + Ɛ$.

Identity rules for regular expressions:

1. $Ø + R = R$
2. $Ø. R = R Ø = Ø$
3. $Ɛ R = R Ɛ = R$
4. $Ø^* = Ɛ$ and $Ɛ^* = Ɛ$
5. $R + R = R$
6. $RR^* = R^* R = R^+$
7. $Ɛ + RR^* = R^*$ and $Ɛ + R^* R = R^*$
8. $(R^*)^* = R^*$
9. $R^* R^* = R^*$

10. $\varepsilon + R^* = R^*$

11. $(R + \varepsilon)^* = R^*$

12. $R^*(\varepsilon + R)^* = (\varepsilon + R)^* R^* = R^*$

13. $R^* R + R = R^* R$

14. $(P + Q)R = PQ + QR$ and $R(P + Q) = RP + RQ$

15. $(P + Q)^* = (P^* Q^*)^* = (P^* + Q^*)^*$

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

17. R is given as, $R = Q + RP$ has unique solution, $R = QP^*$.

This is Arden's theorem.

18. $(P + Q)^* = (P^* + Q) = (P + Q^*)$

Example 16: If r_1 and r_2 are regular expressions denoting languages L_1 and L_2 respectively then which of following is false?

(A) $(r_1)|(r_2)$ is regular expression denoting $L_1 \cup L_2$.

(B) $(r_1)(r_2)$ is regular expression denoting $L_1 . L_2$.

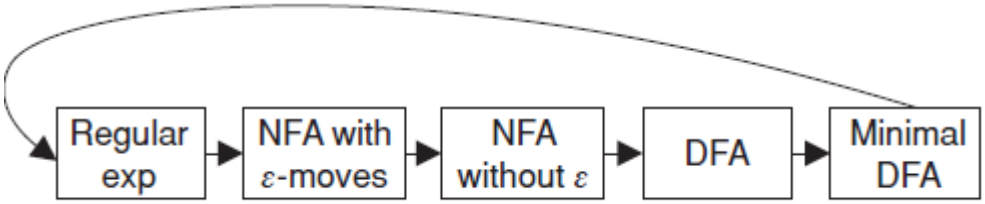
(C) Ø is not a regular expression.

(D) $\{r_1\}^*$ is regular expression denoting L_1^*.

Solution: (C)

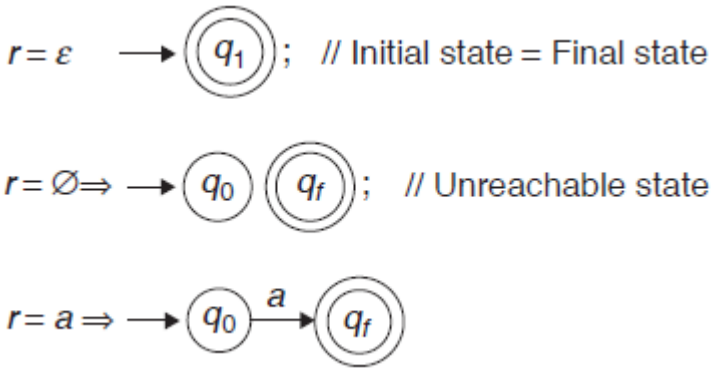
CONSTRUCTING FA FOR GIVEN RE

• Relationship between FA and RE.



Identities:

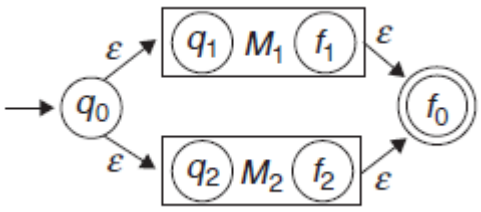
Basis:



Induction:

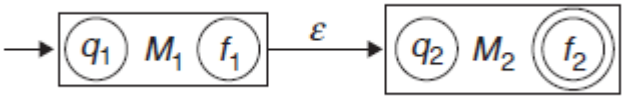
• **Union**: L(r) = L (r_1) +L (r_2) i.e., L (M) = L (M_1) ∪ L (M_2)

Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\})$, $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, \{f_2\})$ with L (M_1) = L (r_1) and L (M_2) = L (r_2), then $M = (Q_1 \cup Q_2 \cup \{q_0, f_0\}, \Sigma_1 \cup \Sigma_2, \delta, q_0, \{f_0\})$



• **Concatenation:**

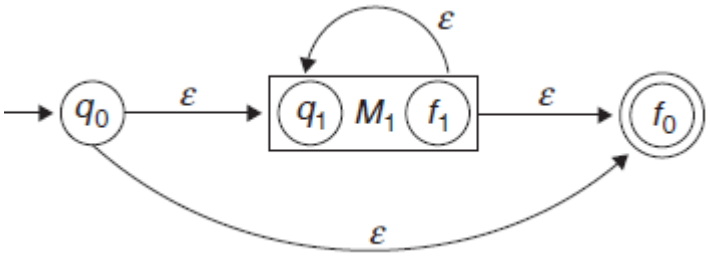
L(r) = L (r_1) . L (r_2) i.e., L (M) = L (M_1) . L (M_2)



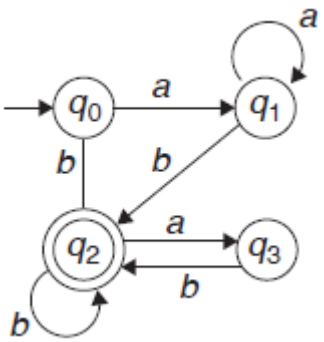
• **Closure:**

L(r) = L(r)* i.e., L (M) = L (M_1)*

Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\})$ then L (M) = $(Q_1 \cup \{q_0, f_0\}, \Sigma_1, \delta, q_0, \{f_0\})$



Example 17: The regular expression generated by the given FA.



(A) $(a + ba^*)\, b^*$ (B) $(aa^* b + bb^*)\, b^*$

(C) $(b + ab^*)\, a^*$ (D) $(ab + ba)^*$

Solution: (B)

q2 is final state which is obtained with input symbol only 'b'. So, (C) or (D) is not true.

In (A) $\rightarrow ba^*$ is not defined in given FA. Instead bb^* is defined.

PUMPING LEMMA FOR REGULAR SETS

Theorem Let 'L' be an arbitrary regular language. Then there exists a positive integer, P with following property:

Given an arbitrary member, w of L having length at least P (i.e., $|w| \geq P$), w can be divided into 3-parts, w =xyz $\exists$

- $|y| \geq 1$ (the middle part is non-empty)
- $|xy| \leq P$ (the first two parts have length atmost P)

• For each, $i \geq 0$, $xy^iz \in L$ (removing or repeating the middle part produces member of L)

Proof Let L be an arbitrary regular language. Then there is a FA, say M that decides L.

Let P be the number of states of M.

Let w be an arbitrary member of L, having length 'n' with $n \geq P$.

Let $q_0, q_1, \ldots q_n$ be states that M on input w. That is, for each i, after reading the first i symbols of w, M is at q_i.

q_0 is initial state of M. Also, since $w \in L$, q_n is a final state of M.

Let $x = w1 \ldots wc$, $y = wc+4 \ldots wd$, $z = wd+1 \ldots wn$. Then:

• $|y| \geq 1$

• $|xy| \leq P$

• M transitions from q_0 to q_c on x.

• M transitions from q_c to q_c on y.

• M transitions from q_c to q_n on z.

Thus, for every $i \geq 0$, M transitions from q_0 to q_n on xy^iz and so, xy^iz is a member of L.

Note:

• Pumping lemma is used to verify that given language is not regular.

• Pumping lemma follows pigeon hole principle.

Example 18: The language, L is defined as:

$L = \{w_1w_2 : w_1, w_2 \in \{a, b\}^*, |w_1| = |w_2|\}$. Is the language regular?

(A) Regular

(B) Not regular

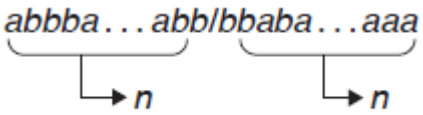
(C) Cannot be determined

(D) None of these

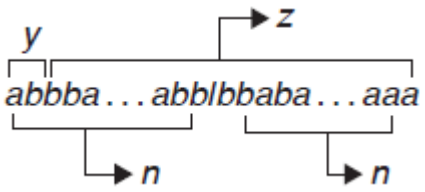
Solution: (A)

Fix pumping length, K =2

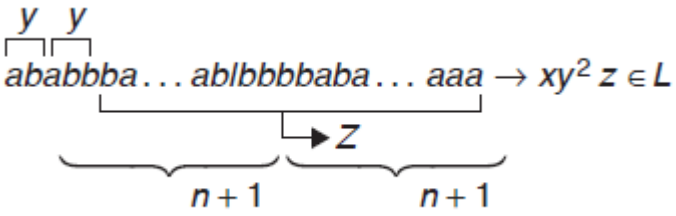
For every proper strings in L, (2n ≥2)



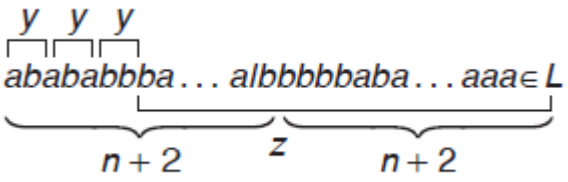
Split in x, y, z with desired properties.



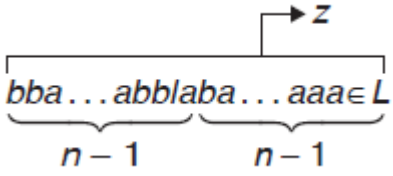
Let x = Ɛ, y =first two symbols, z =rest.



• xy3z:



• $xy^{o}z \rightarrow$



∴For every i ≥0, $xy^i z \in L$. Hence given language is regular.

CLOSURE PROPERTIES OF REGULAR SETS

1. **Union**: If L and M are regular languages, LUM is regular language closed under union.
2. **Concatenation and Kleen closure**: If L and M are regular languages, L.M is regular language and L^* is also regular.
3. **Intersection**: L ∩ M is regular, if L and M are regular languages.
4. **Difference**: L – M contains strings in 'L' but not M, where L and M are regular languages.
5. **Complementation**: The complement of language L is $\Sigma^* - L$.

   Note: Since Σ^* is surely regular, the complement of a regular language is always regular. Where Σ^* is a universal language.
6. **Homomorphism**: If L is a regular language, h is homomorphism on its alphabet then h (L) ={h (w) |w is in L} is also a regular language.

REGULAR GRAMMAR

• **Grammar**: Generative description of a language.

• **Automaton**: Analytical description.

• A grammar is a 4-tuple, G = (V, Σ, R, S) where V:alphabet (variable) (non-terminals)

Σ ⊆ V is set of terminal symbols.

$R \subseteq (V^+ \times V^*)$ is a finite set of production rules.

S ∈ V – Σ is start symbol.

Notation

• Elements of V – Σ : A, B, …

• Elements of Σ : a, b …

• Rules (α , β)∈ R: α →β or $\alpha \xrightarrow[G]{} \beta$

• Start symbol is written as S.

• Empty word: Ɛ

Example 19: The regular expression that describe the language generated by grammar, G = ({S, A, B}, {a, b}, S,{S →Aab, A → Aab|B, B → a}

(A) $(ab)^*$ a (B) aab$(ab)^*$

(C) ab * aa (D) $(a + ba)^*$

Solution: (B)

S →Aab →Aab ab →A ab abab →Bababab

→aababab → aab$(ab)^*$

Union of two Regular languages:

If L_1 and L_2 are two languages then

$L_1 \cup L_2 = \{w/w \in L_1 \text{ or } w \in L_2\}$

The union of two regular languages is also a regular language.

Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, f_1)$

$M_2 = (Q_2, \Sigma, \delta_2, q_2, f_2)$

$M = M_1 U M_2$ can be given as

$M = (Q, \Sigma, \delta, q_0, f)$.

Where $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$

i.e., Q is the Cartesian product of sets Q_1 and Q_2.

Σ is the alphabet, is the same in M_1 and M_2.

$\Sigma = \Sigma_1 U \Sigma_2$.

δ is the transition function given as:

$\delta (r_1, r_2), a = (\delta_1(r_1, a) \delta_2 (r_2, a))$.

q_0 is the pair (q_1, q_2).

F is the set of pairs in which either member is an accept state of M_1 or M_2.

$F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$

TYPES OF GRAMMARS

- Type 0:Unrestricted, recursively enumerable languages.
- Type 1:Context-sensitive grammar.
- Type 2:Context free grammar.
- Type 3:Regular grammar.

Type 0: Recursively enumerable grammar: (Turing Machine) (TM):

Every production rule is of form: $\alpha \rightarrow \beta$, where α and β are in $(V \cup T)^*$, i.e., there can be any strings of terminals and non-terminals (no-restriction).

Type1:Context-sensitive Grammar:(Linear bounded automaton) (LBA):

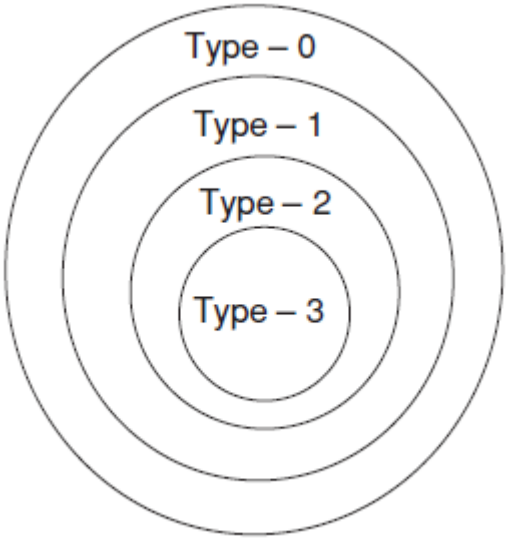
Every production rule is of form, $\alpha \rightarrow \beta$ are in $(V \cup T)^*$ and $\alpha \neq \varepsilon$ and $|\beta| \geq |\alpha|$ i.e., any strings of terminals and nonterminals and length of string that can appear on RHS of production must be greater than or equal to length of string that can appear on LHS of production.

Type2:Context-free grammar: (Push down automaton) (PDA):

Every production rule is of form, $A \rightarrow \alpha$ where α is in $(V \cup T)^*$ i.e., LHS of rule is single non-terminal and RHS can be any string of terminals and non-terminals.

Type 3: Regular grammar: (Finite automaton) (FA):Every production is of form, $A \rightarrow aB$ or $A \rightarrow a$ where A and B $\in$ V and a $\in$ T. That is, LHS of rule is non-terminal and RHS can be terminal (or) terminal followed by non-terminal.

Relationship between types of grammar:



- Regular sets are properly contained in CFL (Context Free Languages).
- The CFL's not containing empty string ε, are properly contained in CSL. (Context sensitive language).
- The CSL's are properly contained in Recursively enumerable languages.
- RG ⊂ CFG ⊂ CSL ⊂ REG

Left-linear Grammar:

All productions have form: A →Bx or A →x

Right-linear Grammar:

All productions have the form: A →xB or A →x.

Note:

- The regular grammars characterize the regular sets i.e., a language is regular if and only if it has a left-linear grammar or if and only if it has a right-linear grammar.
- If L has a regular grammar, then L is a regular set.
- If L is a regular set, then L is generated by some left-linear grammar and by some right-linear grammar.

Arden's theorem: Let P and Q be two regular expressions over Σ. If P does not contain 'Ɛ' then the following equation in R, namely R = Q + RP has a unique solution given by $R = QP^*$.

Arden's Theorem to obtain regular expression from given transition diagram: The following steps are used to find the RE recognized by transition system.

The following assumptions are made regarding the transition system.

(i) The transition graph does not have Ɛ -moves

(ii) It has only one initial state, q_o.

(iii) The states in the transition diagram are $q_o, q_1, q_2, \ldots q_n$.

(iii) Q_i, the regular expression represents the set of strings accepted by a system even though q_i is the final state.

(v) a_{ij} denotes the regular expression representing the set of labels of edges from q_i to q_j. When there is no such edge $a_{ij} = Ø$.

We will get the following set of equations.

$Q_1 = Q_1 \alpha_{11} + Q_2 \alpha_{12} + \ldots Q_n \alpha_{n1} + Ɛ$

$Q_2 = Q_1 \alpha_{12} + Q_2 \alpha_{22} + \ldots Q_n \alpha_{n2}$

:

:

:

$Q_n = Q_1 \alpha_{1n} + Q_2 \alpha_2 n + \ldots Q_n \alpha_{nn}$.

By Repeatedly applying substitutions and Arden's theorem, we can express Q_i in terms of α_{ij}'s.

For getting the set of strings recognized by the transition system, we have to take the union of all Q_i's corresponding to final states.

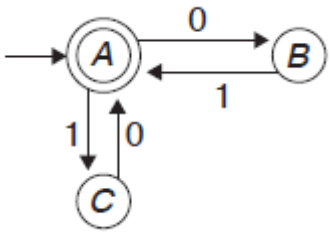
Construction of Regular Grammar from FA

Step I: Associate suitable variables like A, B, C … with states of automata.

Step II: Obtain the productions of the grammar as: If (A, a) = B then add production A →aB to list of productions of grammar, if B is a final state, then add either A →a or B → Ɛ, to list of productions of grammar.

Step III: The variable associated with initial state of automata is start symbol of grammar.

Example 20: Regular grammar generating language accepted by below automata is



(A) A→0B|1C| Ɛ
B→1A
C→0A

(B) A→1B|0C| Ɛ
B→1A
C→0A

(C) A→B|C| Ɛ
B→1
C→0

(D) A→0A|1B| Ɛ
B→1C
C→0A

Solution: (A)

A→0B, A→1C, B→1A, C→0A

∴A is final state, A → Ɛ

∴ A →0B|1C| Ɛ A →0B|1C

B →1A (or) B →1A|1

C →0A C →0A|0

Construction of FA from given regular grammar

Given a regular grammar, G; a regular expression specifying L(G) can be obtained directly as follows:

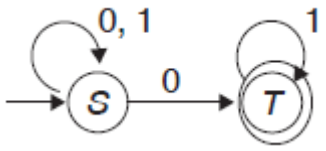
• Replace the '→' symbol in productions of grammar by '=' symbol, to get set of equations.

• Solve the set of equations obtained above to get the value of variable, S, where S is start symbol of grammar, result is regular expression specifying L(G).

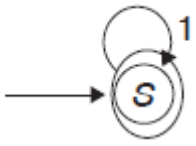
Example 21: The Regular grammar and FA for given regular expression Ø *1*U (0 Ø)* is ___

(A) S→0S|1S|0

T→1T| Ɛ



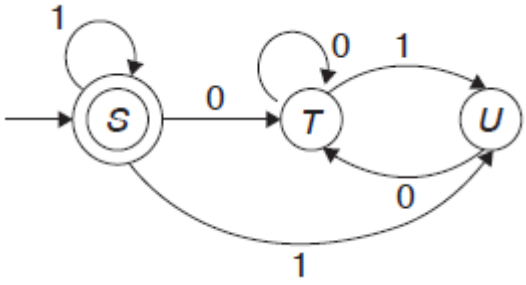
(B) S→1S| Ɛ



(C) S→0T|1S| Ɛ

T→0T|1U| Ɛ

U→0T|1S



(D) Cannot be determined

Solution: (B)

$Ø^* 1^* \cup (0Ø)^* = Ø^*.1^* \cup Ø^* = Ɛ.1^* \cup Ɛ = 1^*$.