Automata Theory

Automata Theory is a branch of computer science that deals with designing abstract self-propelled computing devices that follow a predetermined sequence of operations automatically. An automaton with a finite number of states is called a **Finite Automaton**. This is a brief and concise tutorial that introduces the fundamental concepts of Finite Automata, Regular Languages, and Pushdown Automata before moving onto Turing machines and Decidability.

Automata – What is it?

The term "Automata" is derived from the Greek word "αὐτόματα" which means "self-acting". An automaton (Automata in plural) is an abstract self-propelled computing device which follows a predetermined sequence of operations automatically.

An automaton with a finite number of states is called a **Finite Automaton** (FA) or **Finite State Machine** (FSM).

Formal definition of a Finite Automaton

An automaton can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where –

- **Q** is a finite set of states.
- **Σ** is a finite set of symbols, called the **alphabet** of the automaton.
- **δ** is the transition function.
- **q_0** is the initial state from where any input is processed ($q_0 \in Q$).
- **F** is a set of final state/states of Q ($F \subseteq Q$).

Related Terminologies

Alphabet

- **Definition** – An **alphabet** is any finite set of symbols.
- **Example** – $\Sigma = \{a, b, c, d\}$ is an **alphabet set** where 'a', 'b', 'c', and 'd' are **alphabets**.

String

- **Definition** – A **string** is a finite sequence of symbols taken from Σ.
- **Example** – 'cabcad' is a valid string on the alphabet set $\Sigma = \{a, b, c,d\}$

Length of a String

- **Definition** – It is the number of symbols present in a string. (Denoted by **|S|**).
- **Examples** –
    - If S='cabcad', |S|= 6
    - If |S|= 0, it is called an **empty string** (Denoted by **λ** or **ε**)

Kleene Star

- **Definition** – The set $\mathbf{\Sigma^*}$ is the infinite set of all possible strings of all possible lengths over **Σ** including **λ**.
- **Representation** – $\Sigma^* = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2 \cup \ldots\ldots$
- **Example** – If $\Sigma = \{a, b\}$, $\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, \ldots\ldots\ldots..\}$

Kleene Closure/Plus

- **Definition** – The set $\mathbf{\Sigma^+}$ is the infinite set of all possible strings of all possible lengths over **Σ** excluding **λ**.
- **Representation** – $\Sigma^+ = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2 \cup \ldots\ldots$

  $\Sigma^+ = \Sigma^* - \{ \lambda \}$

- **Example** – If Σ = { a, b } , Σ^+ ={ a, b, aa, ab, ba, bb,…….}

Language

- **Definition** – A language is a subset of Σ^* for some alphabet Σ. It can be finite or infinite.
- **Example** – If the language takes all possible strings of length 2 over Σ = {a, b}, then L = { ab, bb, ba, bb}

Finite Automaton can be classified into two types –

- Deterministic Finite Automaton (DFA)
- Non-deterministic Finite Automaton (NDFA / NFA)

Deterministic Finite Automaton (DFA)

In DFA, for each input symbol, one can determine the state to which the machine will move. Hence, it is called **Deterministic Automaton.** As it has a finite number of states, the machine is called **Deterministic Finite Machine**or **Deterministic Finite Automaton.**

Formal Definition of a DFA

A DFA can be represented by a 5-tuple (Q, Σ, δ, q_0, F) where –

- **Q** is a finite set of states.
- **Σ** is a finite set of symbols called the alphabet.
- **δ** is the transition function where δ: Q × Σ → Q
- **q_0** is the initial state from where any input is processed (q0 ∈ Q).
- **F** is a set of final state/states of Q (F ⊆ Q).

Graphical Representation of a DFA

A DFA is represented by digraphs called **state diagram.**

- The vertices represent the states.
- The arcs labeled with an input alphabet show the transitions.
- The initial state is denoted by an empty single incoming arc.
- The final state is indicated by double circles.

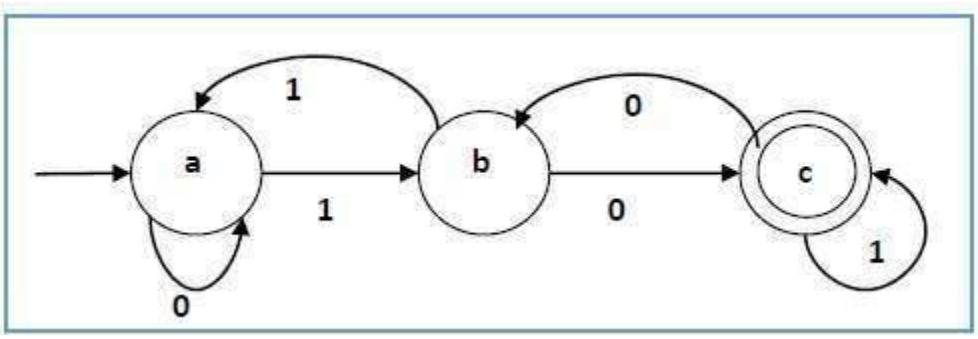
Example

Let a deterministic finite automaton be →

- Q = {a, b, c},
- Σ = {0, 1},
- q_0={a},
- F={c}, and
- Transition function δ as shown by the following table –

Present State	Next State for Input 0	Next State for Input 1
A	a	b
B	c	a
C	b	c

Its graphical representation would be as follows –



Non-deterministic Finite Automaton

In NDFA, for a particular input symbol, the machine can move to any combination of the states in the machine. In other words, the exact state to which the machine moves cannot be determined. Hence, it is called **Non-deterministic Automaton.** As it has finite number of states, the machine is called **Non-deterministic Finite Machine** or **Non-deterministic Finite Automaton**.

Formal Definition of an NDFA

An NDFA can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where –

- **Q** is a finite set of states.
- **Σ** is a finite set of symbols called the alphabets.
- **δ** is the transition function where $\delta: Q \times \{\Sigma \cup \varepsilon\} \rightarrow 2Q$ (Here the power set of Q (2^Q) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states)
- **q_0** is the initial state from where any input is processed ($q_0 \in Q$).
- **F** is a set of final state/states of Q ($F \subseteq Q$).

Graphical Representation of an NDFA – (same as DFA)

An NDFA is represented by digraphs called state diagram.

- The vertices represent the states.
- The arcs labeled with an input alphabet show the transitions.
- The initial state is denoted by an empty single incoming arc.
- The final state is indicated by double circles.

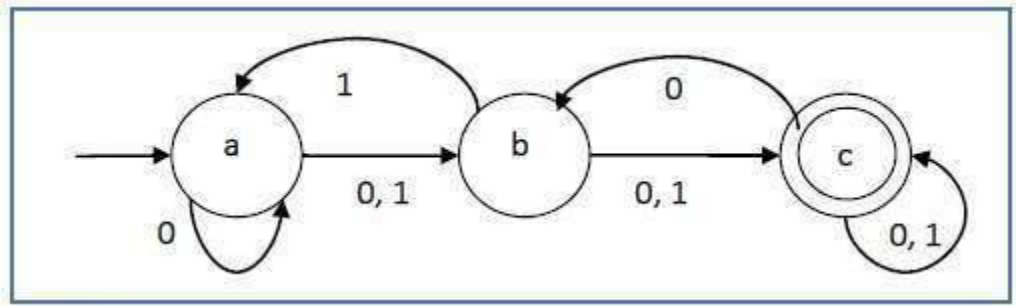
Example

Let a non-deterministic finite automaton be →

- Q = {a, b, c}
- Σ = {0, 1}
- q_0 = {a}
- F={c}
- The transition function δ as shown below –

Present State	Next State for Input 0	Next State for Input 1
a	a, b	b
b	c	a, c
c	b, c	c

Its graphical representation would be as follows –



Acceptors, Classifiers, and Transducers

Acceptor (Recognizer)

An automaton that computes a Boolean function is called an **acceptor.** All the states of an acceptor is either accepting or rejecting the inputs given to it.

Classifier

A **classifier** has more than two final states and it gives a single output when it terminates.

Transducer

An automaton that produces outputs based on current input and/or previous state is called a **transducer.** Transducers can be of two types –

- **Mealy Machine** – The output depends only on the current state.
- **Moore Machine** – The output depends both on the current input as well as the current state.

Acceptability by DFA and NDFA

A string is accepted by a DFA/NDFA iff the DFA/NDFA starting at the initial state ends in an accepting state (any of the final states) after reading the string wholly.

A string S is accepted by a DFA/NDFA $(Q, \Sigma, \delta, q_0, F)$, iff

$$\delta^*(q_0, S) \in F$$

The language **L** accepted by DFA/NDFA is

$$\{S \mid S \in \Sigma^* \text{ and } \delta^*(q_0, S) \in F\}$$

A string S' is not accepted by a DFA/NDFA $(Q, \Sigma, \delta, q_0, F)$, iff

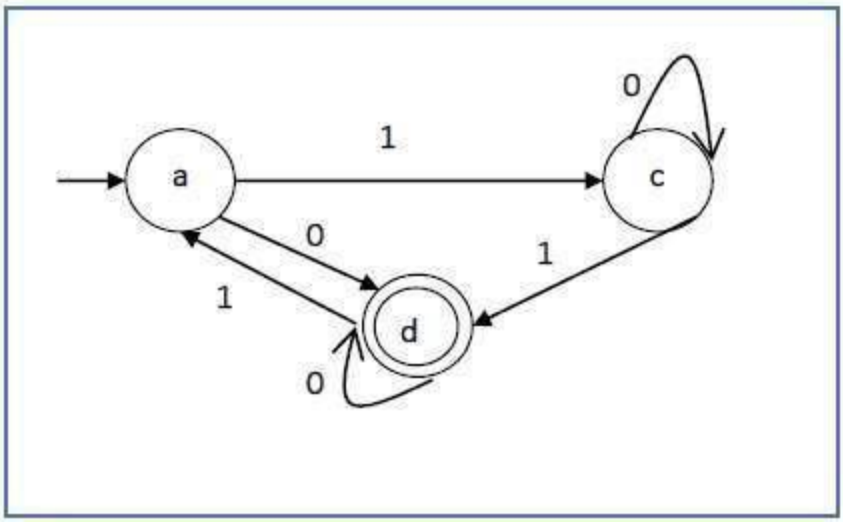
$$\delta^*(q_0, S') \notin F$$

The language L' not accepted by DFA/NDFA (Complement of accepted language L) is

$$\{S \mid S \in \Sigma^* \text{ and } \delta^*(q_0, S) \notin F\}$$

Example

Let us consider the DFA shown in Figure 1.3. From the DFA, the acceptable strings can be derived.



Strings accepted by the above DFA – {0, 00, 11, 010, 101, ...}

Strings not accepted by the above DFA – {1, 011, 111, ...}

NDFA to DFA Conversion

Problem Statement

Let $\mathbf{X = (Q_x, \Sigma, \delta_x, q_0, F_x)}$ be an NDFA which accepts the language L(X). We have to design an equivalent DFA $\mathbf{Y = (Q_y, \Sigma, \delta_y, q_0, F_y)}$ such that **L(Y) = L(X).** The following procedure converts the NDFA to its equivalent DFA –

Algorithm

Input: An NDFA

Output: equivalent DFA

Step 1 Create state table from the given NDFA.

Step 2 Create a blank state table under possible input alphabets for the equivalent DFA.

Step 3 Mark the start state of the DFA by q_0 (Same as the NDFA).

Step 4 Find out the combination of States $\{Q_0, Q_1, \ldots, Q_n\}$ for each possible input alphabet.

Step 5 Each time we generate a new DFA state under the input alphabet columns, we have to apply step 4 again, otherwise go to step 6.

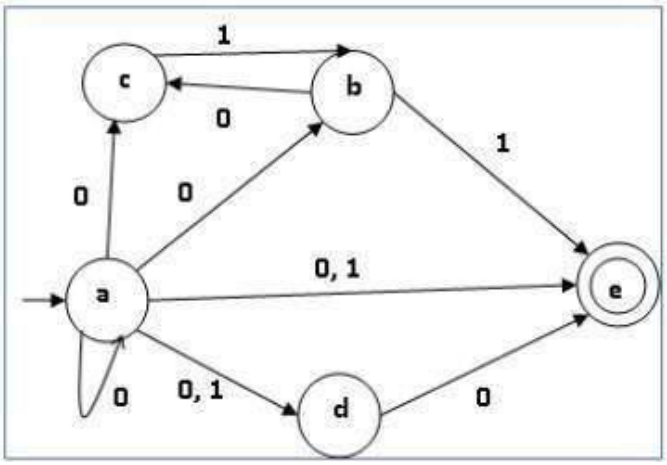
Step 6 The states which contain any of the final states of the NDFA are the final states of the equivalent DFA.

Example

The NDFA table is as follows –

q	δ(q,0)	δ(q,1)
a	{a,b,c,d,e}	{d,e}
b	{c}	{e}
c	∅	{b}
d	{e}	∅
e	∅	∅

Let us consider the NDFA shown in the figure below.

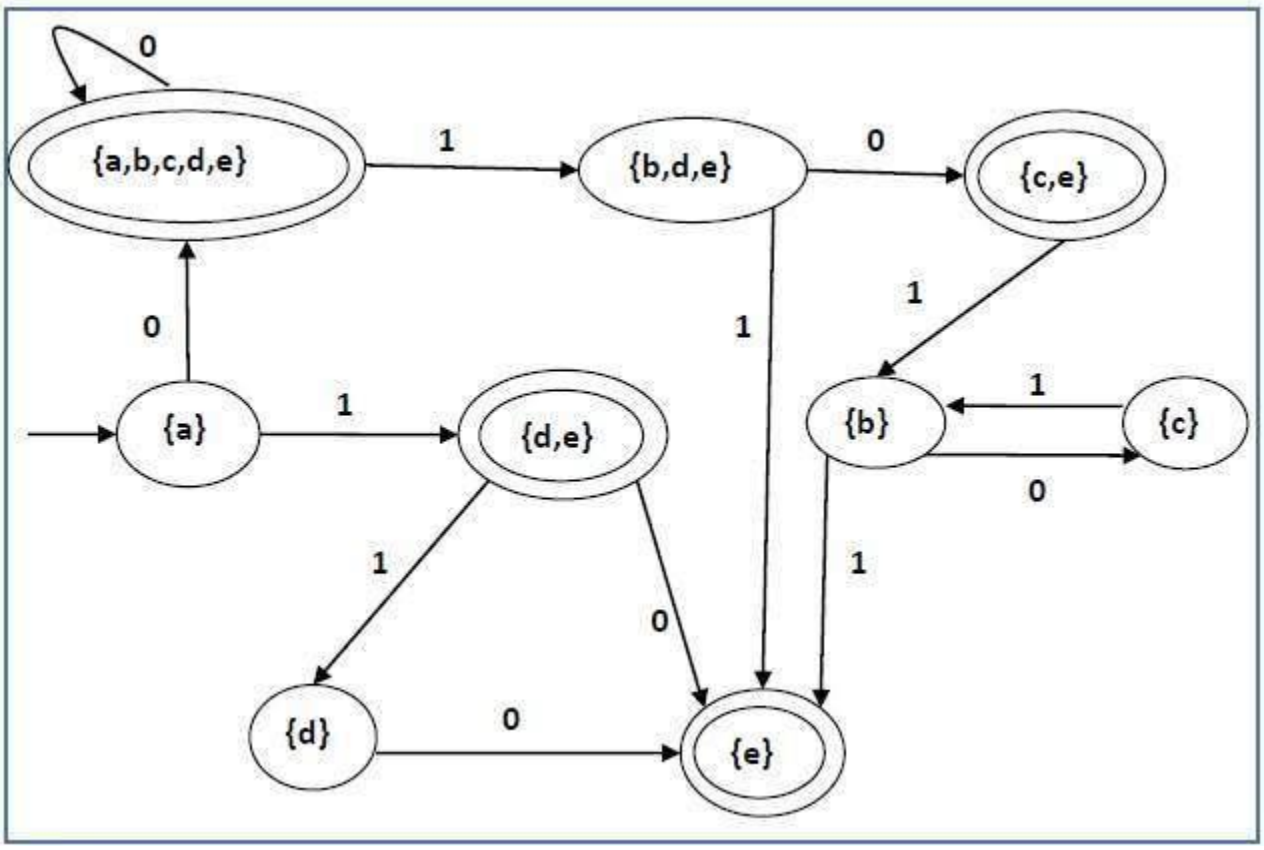


Using above algorithm, we find its equivalent DFA. The state table of the DFA is shown in below.

q	δ(q,0)	δ(q,1)
a	{a,b,c,d,e}	{d,e}
{a,b,c,d,e}	{a,b,c,d,e}	{b,d,e}
{d,e}	E	D
{b,d,e}	{c,e}	E
e	∅	∅
d	E	∅
{c,e}	∅	B
b	C	E

c	∅	B

The state diagram of the DFA is as follows −



DFA Minimization

DFA Minimization using Myphill-Nerode Theorem

Algorithm

Input: DFA

Output: Minimized DFA

Step 1 Draw a table for all pairs of states (Q_i, Q_j) not necessarily connected directly [All are unmarked initially]

Step 2 Consider every state pair (Q_i, Q_j) in the DFA where $Q_i \in F$ and $Q_j \notin F$ or vice versa and mark them. [Here F is the set of final states].

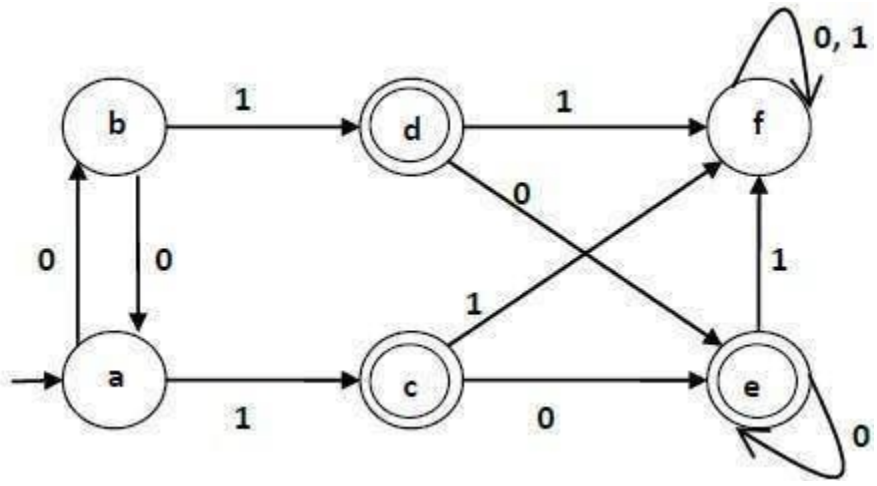
Step 3 Repeat this step until we cannot mark anymore states −

If there is an unmarked pair (Q_i, Q_j), mark it if the pair $\{\delta(Q_i, A), \delta(Q_i, A)\}$ is marked for some input alphabet.

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

Example

Let us use above algorithm to minimize the DFA shown below.



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

	a	b	c	d	e	f
a						
b						
c						
d						
e						
f						

Step 2 – We mark the state pairs –

	a	b	c	d	e	f
a						
b						
c	✓	✓				
d	✓	✓				
e	✓	✓				

f			✓	✓	✓	

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

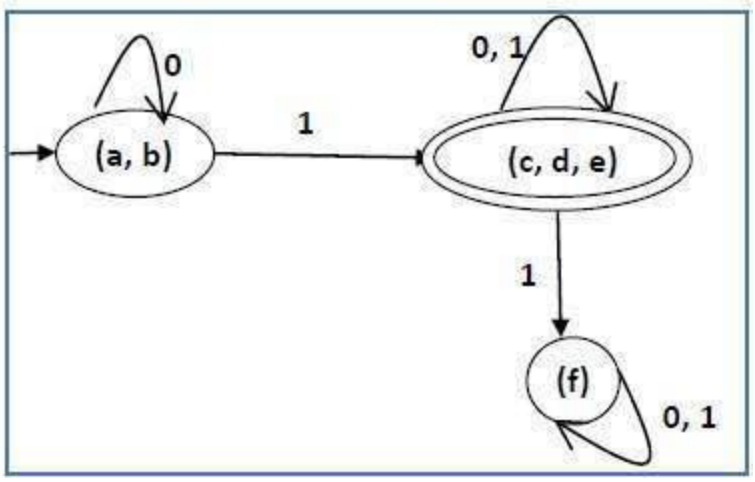
	a	**b**	**c**	**d**	**e**	**f**
a						
b						
c	✓	✓				
d	✓	✓				
e	✓	✓				
f	✓	✓	✓	✓	✓	

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

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

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

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



DFA Minimization using Equivalence Theorem

If X and Y are two states in a DFA, we can combine these two states into {X, Y} if they are not distinguishable. Two states are distinguishable, if there is at least one string S, such that one of δ (X, S) and δ (Y, S) is accepting and another is not accepting. Hence, a DFA is minimal if and only if all the states are distinguishable.

Algorithm

Step 1 All the states **Q** are divided in two partitions – **final states** and **non-final states** and are denoted by $\mathbf{P_0}$. All the states in a partition are 0^{th}equivalent. Take a counter **k** and initialize it with 0.

Step 2 Increment k by 1. For each partition in P_k, divide the states in P_k into two partitions if they are k-distinguishable. Two states within this partition X and Y are k-distinguishable if there is an input **S** such that **δ(X, S)** and**δ(Y, S)** are (k-1)-distinguishable.

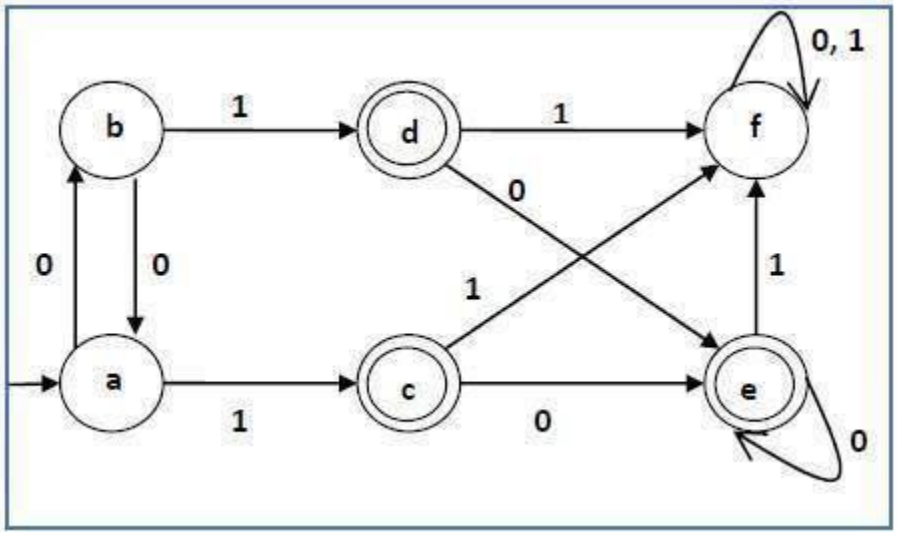
Step 3 If $P_k \neq P_{k-1}$, repeat Step 2, otherwise go to Step 4.

Step 4 Combine k^{th} equivalent sets and make them the new states of the reduced DFA.

Example

Let us consider the following DFA –

q	δ(q,0)	δ(q,1)
a	b	c
b	a	d
c	e	f
d	e	f
e	e	f
f	f	f



Let us apply above algorithm to the above DFA –

- P_0 = {(c,d,e), (a,b,f)}
- P_1 = {(c,d,e), (a,b),(f)}
- P_2 = {(c,d,e), (a,b),(f)}

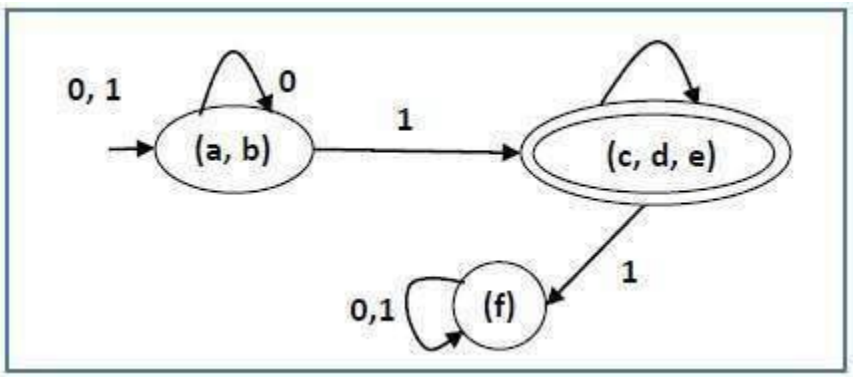
Hence, $P_1 = P_2$.

There are three states in the reduced DFA. The reduced DFA is as follows –

The State table of DFA is as follows –

Q	δ(q,0)	δ(q,1)
(a, b)	(a, b)	(c,d,e)
(c,d,e)	(c,d,e)	(f)
(f)	(f)	(f)

Its graphical representation would be as follows –



4/03/2021

Formal Language And Automata Theory

Basic Definitions or Terminology

1) **Symbol** :- Symbol is a character.

eg: a, b, c .... z Alpha character

0, 1, 2 .... 9 Numeric character

+, −, % .... Special character

2) **Alphabet** :- Alphabet is a finite set of symbols.

Alphabet is denoted by Σ.

eg: $\Sigma = \{0, 1\}$ set of binary alphabets

$\Sigma = \{a, b, c \ldots z\}$ set of lowercase alphabets.

$\Sigma = \{a\text{-}z, A\text{-}Z, 0\text{-}9\}$

$\Sigma = \{+, -, *, / \ldots\}$

3) **String** :- A string or a word is a finite sequence of alphabets.

A string is denoted by w.

eg: 1) abab is a string or word over alphabet

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

2) 101100 is a string or word over alphabet

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

Symbol → alphabet → string

Scanned with CamScanner

4) Empty string :- The Empty string is the string with Zero occurence of a symbol (no symbol).
It is denoted by 'ϵ' or 'λ'

eg: $\{\epsilon\}$

5) Length of string :- The length of the string is the number of symbols in a word.
It is denoted by $|w|$.

eg: 010101 is a string over alphabet $\Sigma = \{0,1\}$

$|w| = 6$

6) Concatenation of strings :- The concatenation of two strings w and v is obtained by appending the symbols of v to the right end of w, that is if

$w = a_1 a_2 \ldots a_n$

$v = b_1 b_2 \ldots b_m$

then the concatenation of string w & v is denoted by wv.

$wv = a_1 a_2 \ldots a_n b_1 b_2 \ldots b_m$

eg: w = abaaa and v = cdddcd

wv = abaaacdddcd

7) Reverse of string :- The reverse of a string is obtained by writing the symbols in reverse order.
If w is a string, then its reverse w^R is

$w^R = a_n \ldots a_2 a_1$

eg: w = abaabab

w^R = babaaba

Scanned with CamScanner

Language :- A set of string all of which are choosen from some Σ^*, where Σ is a particular alphabet is called language.

eg: Collection of legal English words is a Set of strings over the alphabet.

Closure - Multiple occurrence of a symbol.

Kleen's closure :- If Σ is the set of alphabets, then there is a language in which any string of letters from Σ is a word, even the null string. we call this language closure of the alphabet.

- It is denoted by * (asterick) after the name of the alphabet is Σ^*. This notation is also known as kleen's star.

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \dots\dots$$

eg: Let $\Sigma = \{a, b\}$ then

$$\Sigma^* = \{\epsilon\} \cup \{a, b\} \cup \{aa, ab, ba \dots\}$$

Positive closure :- one or more instances of Σ.

- denoted by '+'
- Σ^+ = the set of non-empty strings from Σ.

$$\Sigma^+ = \Sigma^* - \{\epsilon\}$$

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \dots\dots$$

eg: Let $\Sigma = \{a, b\}$ then

$$\Sigma^+ = \{aa, ab, ba \dots\dots\}$$

Scanned with CamScanner

Describe the Language :-

1) Strings of length exactly 2 over $\Sigma = \{a, b\}$

Soln Given: $\Sigma = \{a, b\}$

The language L_1 is described as

$L = \{aa, ab, ba, bb\}$

2) Strings of length atmost 3 over $\Sigma = \{a, b\}$

Soln Given $\Sigma = \{a, b\}$

The language L_2 is described as

$L = \{\epsilon, a, b, aa, bb, ab, ba, aaa, bbb, aba, bab, aab, bba, abb, baa\}$

3) Strings over Σ of length atleast 2, $\Sigma = \{a, b\}$

Soln Given $\Sigma = \{a, b\}$

The language L_3 is described as

$L = \{aa, ab, ba, bb, aaa, aab, aba, abb, bab, baa, bba, bbb\}$

4) Strings containing b over $\Sigma = \{a, b\}$

Soln Given $\Sigma = \{a, b\}$

The language L_4 is described as

$L = \{b, ba, ab, bba, aab, aba, abb, baaa \ldots\}$

5) Strings beginning with a over $\Sigma = \{a, b\}$

Soln Given $\Sigma = \{a, b\}$

The Language L_5 is described as

$L = \{a, aa, ab, aaa, aab, aba, abb, aaaa \ldots\}$

Scanned with CamScanner

6) Strings over $\Sigma = \{a\}$ of length multiple of 3.

Soln Given $\Sigma = \{a\}$

$L = \{\epsilon, aaa, aaaaaa, aaaaaaaaa, \ldots\}$

7) Strings over $\Sigma = \{a, b\}$ of length divisible by 3.

Soln Given $\Sigma = \{a, b\}$

$L = \{\epsilon, aaa, aab, aba, abb \ldots\}$

8) Strings over $\{a, b\}^*$ containing two consecutive a's.

Soln Given $\Sigma = \{a, b\}$

$L = \{aa, aaa, aab, baa, aaaa, \ldots\}$

9) Strings over $\{a, b\}$ every 'a' is followed by two b's.

Soln Given $\Sigma = \{a, b\}$

$L = \{abb, babb, abbb, abbabb, bbabb, \ldots\}$

10) Binary string as integer divisible by 3 over $\{0, 1\}$.

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

$L = \{0, 11, 110, 1001, 1100, 1111\}$

11) String containing two consecutive zeros or two consecutive one over $\Sigma = \{0, 1\}$

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

$L = \{00, 11, 000, 001, 011, 100, 110, 111, 0000, 0001, 0010, \ldots\}$

12) $L = \{0^n 1^{2n} \mid n \geq 1\}$

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

$L = \{011, 001111, 000111111, \ldots\}$

13) $L = \{0^n 1 \mid n > 0\}$

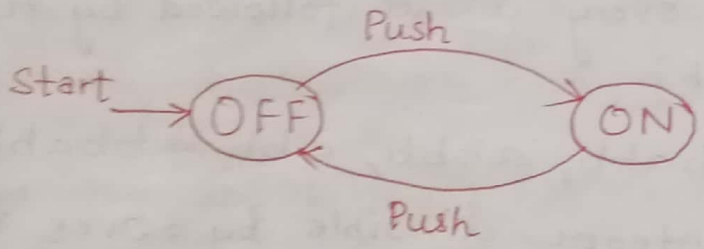
Soln Given $\Sigma = \{0, 1\}$

$L = \{1, 01, 001, 0001, 00001, \ldots\}$

Scanned with CamScanner

Finite Automata

- Finite Automata is a State Machine that takes a Strings of symbols as input and changes its state accordingly.
- Finite Automation is called 'finite' because number of possible states and number of letter in the alphabet are both finite and automation because the change of the state is totally governed by the input.
- Example Incase of Electric Switch only two Situation is possible 'ON' and 'OFF'.



Representation

State (q)

Initial state $\rightarrow (q)$

Final / Accepting state $((q))$

Transition $\rightarrow (q_i) \xrightarrow{a} (q_j)$, $(q_i) \xrightarrow{a,b} (q_j)$

There are two types of finite automata

1) Deterministic finite state Automata $(i) \xrightarrow{a} (j)$
2) Non-Deterministic finite State Automata. $(i) \xrightarrow{a} (j)$, $(i) \xrightarrow{a} (k)$

Scanned with CamScanner

Deterministic finite Automata - (DFA)

A Deterministic finite Automata is a 5-tuple defined by $M = (Q, \Sigma, \delta, q_0, F)$

Q :- Is a non-empty finite set of states.

F :- Is a non-empty finite set of input symbols or alphabets.

q_0 :- Is a starting state / An initial state $q_0 \in Q$.

F :- Is a non-empty set of final states or accepting States $F \subseteq Q$

δ :- Is a function called transition function that takes two argument a state and an input symbol, It returns a single state.

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

Features :-

1) Transition :- from Every state there must be a transition for every input alphabets.

2) Multiple Transition :- More than 1 transition for same i/p alphabet symbol is not permitted.

3) ϵ-transitions :- ϵ-transitions is not permitted.

Scanned with CamScanner

eg:-

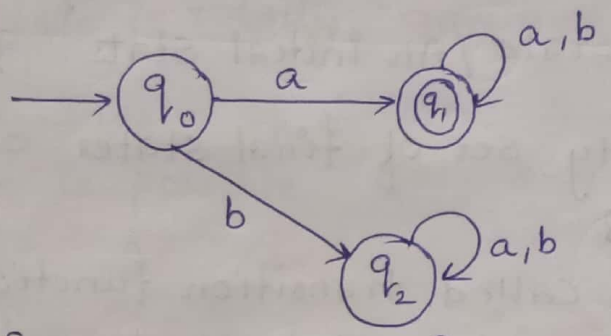
1) Create/Design a DFA for Language L over {a,b} that Contain a set of strings that begin with a.

Soln

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

$$L = \{a, ab, abb, abab \ldots\}$$

DFA is



$$M = \{Q, \Sigma, \delta, q_0, F\}$$

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

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

$$q_0 = \{q_0\}$$

$$F = \{q_1\}$$

δ is defined by

$\delta(q_0, a) = q_1$ $\quad$ $\delta(q_0, b) = q_2$

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

$\delta(q_2, a) = q_2$ $\quad$ $\delta(q_2, b) = q_2$

δ is defined by

State	O/P	
	i/p=a	i/p=b
→ q_0	q_1	q_2
q_1	q_1	q_1
* q_2	q_2	q_2

→ initial state

* final state

Scanned with CamScanner

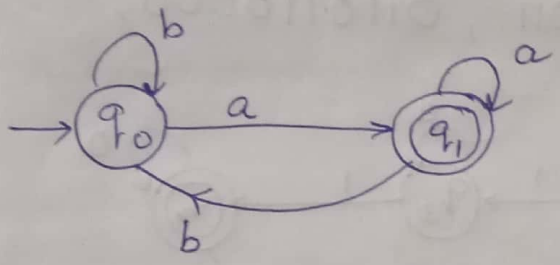
2) Design a DFA for the language L over {a,b} that contain set of strings which are ending with a.

Soln

$$\Sigma = \{a,b\}^*$$

$$L = \{a, ba, baa, aba, \ldots\ldots\}$$

DFA is



$M = \{Q, \Sigma, \delta, q_0, F\}$

$Q = \{q_0, q_1\}$

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

$q_0 = \{q_0\}$

$F = \{q_1\}$

δ is defined by

$\delta(q_0, a) = q_1$

$\delta(q_1, a) = q_1$

$\delta(q_0, b) = q_0$

$\delta(q_1, b) = q_0$

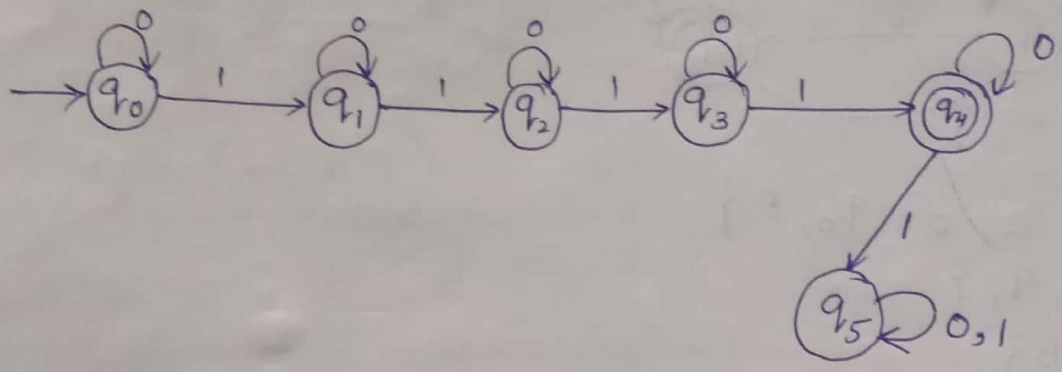
State	O/P	
	i/p=a	i/p=b
$\rightarrow q_0$	q_1	q_0
$* q_1$	q_1	q_0

Scanned with CamScanner

3) Design a DFA which accepts set of strings over alphabet containing exactly four 1's in every string $\Sigma = \{0,1\}$

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

$L = \{1111, 01010111, 011011110000, \ldots\}$



$M = \{Q, \Sigma, \delta, q_0, F\}$

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

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

$q_0 = \{q_0\}$

$F = \{q_4\}$

δ is defined by

$\delta(q_0,0) = q_0$ $\quad \delta(q_0,1) = q_1$

$\delta(q_1,0) = q_1$ $\quad \delta(q_1,1) = q_2$

$\delta(q_2,0) = q_2$ $\quad \delta(q_2,1) = q_3$

$\delta(q_3,0) = q_3$ $\quad \delta(q_3,1) = q_4$

$\delta(q_4,0) = q_4$ $\quad \delta(q_4,1) = q_5$

$\delta(q_5,0) = q_5$ $\quad \delta(q_5,1) = q_5$

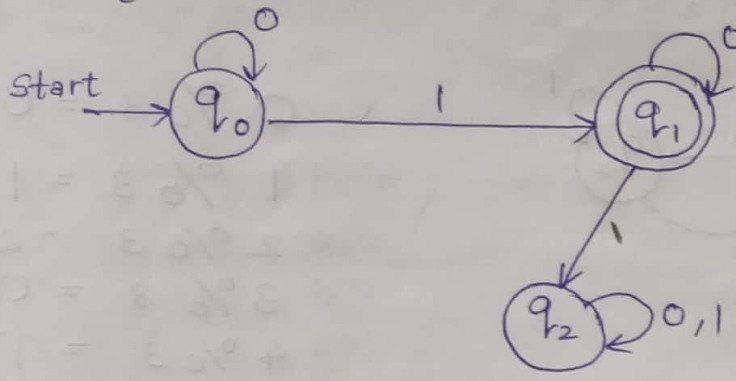
State	i/p	
	0	1
→ q_0	q_0	q_1
q_1	q_1	q_2
q_2	q_2	q_3
q_3	q_3	q_4
* q_4	q_4	q_5
q_5	q_5	q_5

Scanned with CamScanner

4) Design a Finitute automata that accepts strings containing exactly 1 over alphabet $\{0,1\}$.

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

$L = \{1, 0001, 01000, 0001000, \ldots\}$



$M = \{Q, \Sigma, \delta, q_0, F\}$

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

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

$q_0 = \{q_0\}$

$F = \{q_1\}$

Transition Table

State	I/P	
	0	1
→ q_0	q_0	q_1
q_1	q_1	q_2
* q_2	q_2	q_2

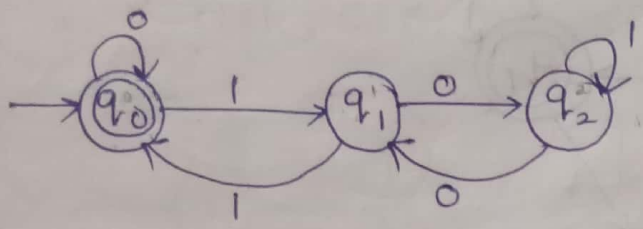
Scanned with CamScanner

5) Design a DFA for the language $\{0, 1\}$ that contains all strings which are divisible by 3.

Soln

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

$$L = \{0, 11, 110, 1001, \ldots\}$$



000 0 % 3 = 0 → q_0
001 1 % 3 = 1 → q_1
010 2 % 3 = 2 → q_2
011 3 % 3 = 0 → q_0
100 4 % 3 = 1 → q_1
101 5 % 3 = 2 → q_2

$$M = \{Q, \Sigma, \delta, q_0, F\}$$

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

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

$$q_0 = \{q_0\}$$

$$F = \{q_0\}$$

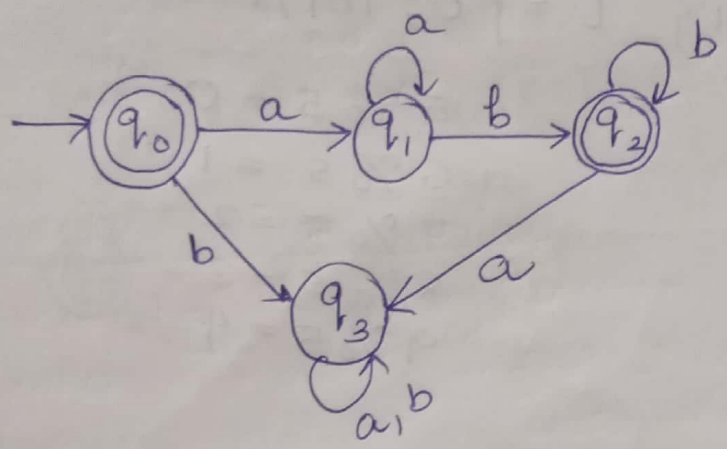
Transition Table

State	I/P	
	0	1
→ q_0	q_0	q_1
q_1	q_2	q_0
* q_2	q_1	q_2

Scanned with CamScanner

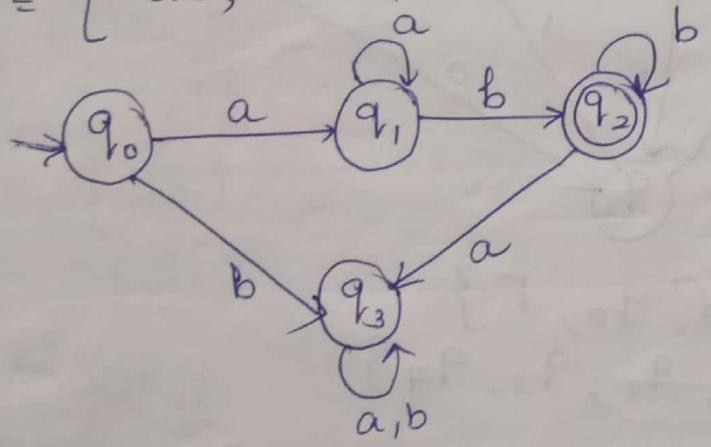
7) $L = \{ a^n b^m \mid n, m \geq 0 \}$ $a^0 b^0 = \epsilon$

$L = \{ ab, aabb, aaab, abb \ldots \}$

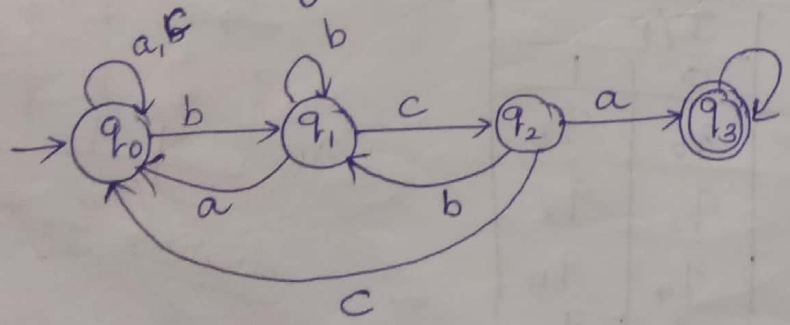


10) Design a DFA $L = \{ a^n b^m \mid n, m \geq 1 \}$

$L = \{ ab, aab, abb, \ldots \}$



11) Design a DFA for the language L over alphabet $L = \{ \omega \mid \omega \in \{a, b, c\}$ and having 'bca' as a substring $\}$

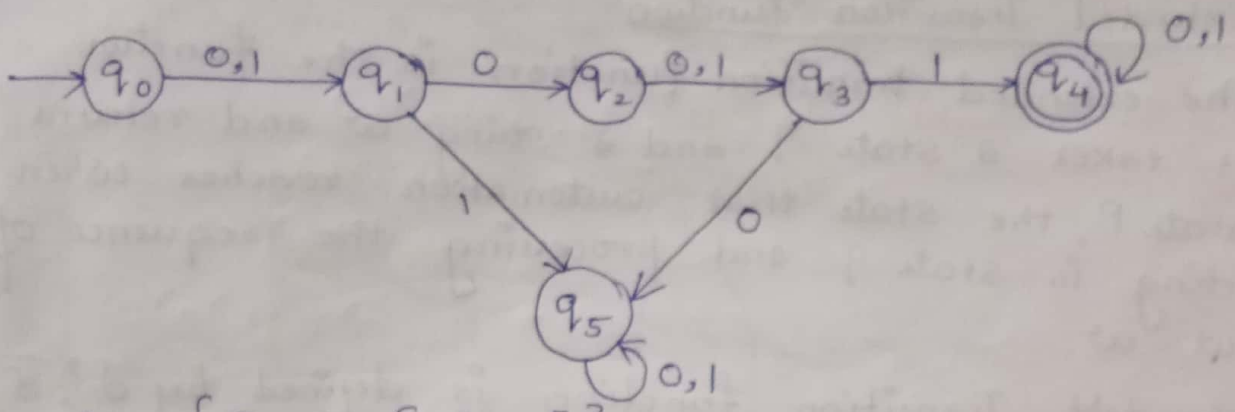


Scanned with CamScanner

12) Design a finite Automata which accepts the language

$L = \{ w \in (0,1)^* \mid$ where Second symbol of w is '0' and fourth input symbol is 1 $\}$

$L = \{0001, 1011, 10110000, \ldots\}$



$M = \{Q, \Sigma, \delta, q_0, F\}$

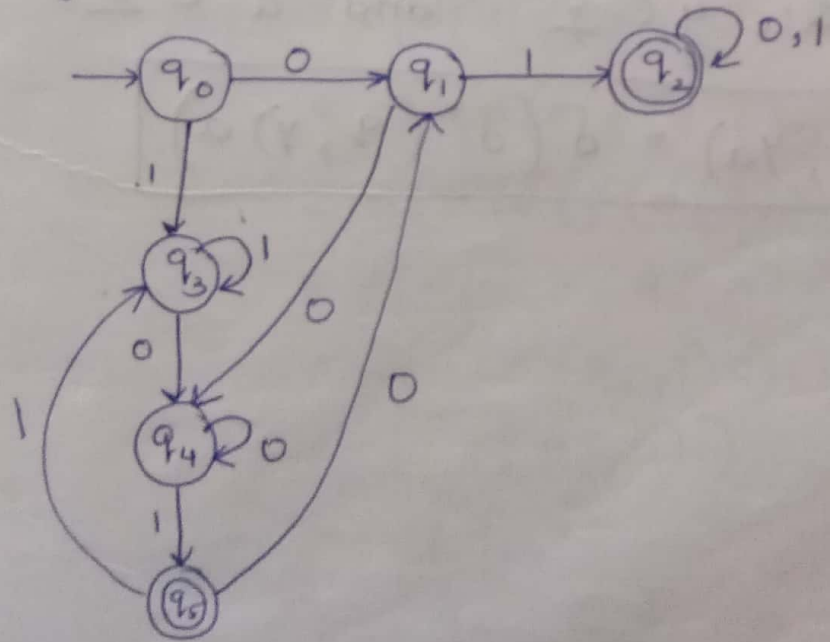
$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

$F = \{q_4\}$

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

13) Design a DFA over alphabet $\Sigma = \{0,1\}$ which accepts the set of strings either start with 01 or ends with 01

$L = \{01, 011111, 01000, 101, 0011101, 11101, \ldots\}$



Scanned with CamScanner

Transition Function

$\delta(q_i, a) = q_j$ → next state

- q_i: initial state
- a: input alphabet

Extended Transition function

- The extended transition function is the function that takes a state q and a string w and returns a state P, the state that automation reaches when starting in state q and processing the sequence of input w.
- Extended Transition function is defined by $\delta^*, \bar{\delta}, \hat{\delta}$
- Let $M = \{Q, \Sigma, \delta, q_s, F\}$ be a finite automaton, then we define the extended transition function

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

as follows

(i) for any $q \in Q$, $\delta^*(q, \epsilon) = q$

(ii) for any $q \in Q$, $y \in \Sigma^*$ and $a \in \Sigma$

$$\delta^*(q, ya) = \delta(\delta^*(q, y)\, a)$$

Scanned with CamScanner

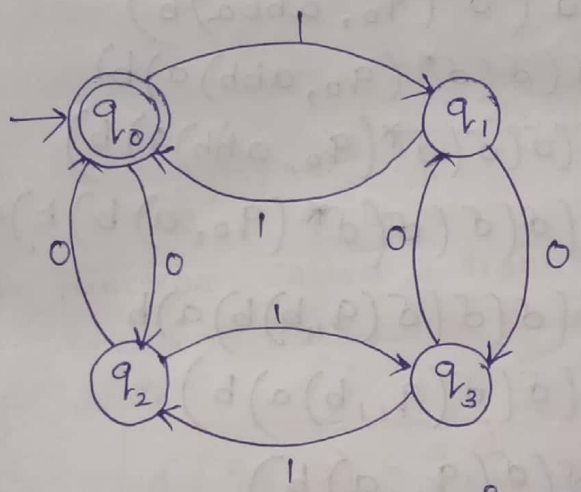
Design a DFA which accepts the language

$L = \{w \mid w$ has both an even number of 0's and an even number of 1's over alphabet $\Sigma = \{0,1\}\}$

Soln

Let q_0 be the initial state, since '0' is even so at initial state number of 0's and number of 1's both are even so clearly q_0 will be final state.

00, 11,



Check whether string 110101 is accepted or not.

$\delta^*(q_0, 110101) = \delta(\delta^*(q_0, 11010)1)$

$= \delta(\delta(\delta^*(q_0, 1101)0)1)$

$= \delta(\delta(\delta(\delta^*(q_0, 110)1)0)1)$

$= \delta(\delta(\delta(\delta(\delta^*(q_0, 11)0)1)0)1)$

$= \delta(\delta(\delta(\delta(\delta(\delta(q_0,1)1)0)1)0)1)$

$= \delta(\delta(\delta(\delta(\delta(q_1,1)\ 0)1)0)1)$

$= \delta(\delta(\delta(\delta(q_0,0)1)0)1)$

$= \delta(\delta(\delta(q_2,1)0)1)$

$= \delta(\delta(q_3,0)1)$

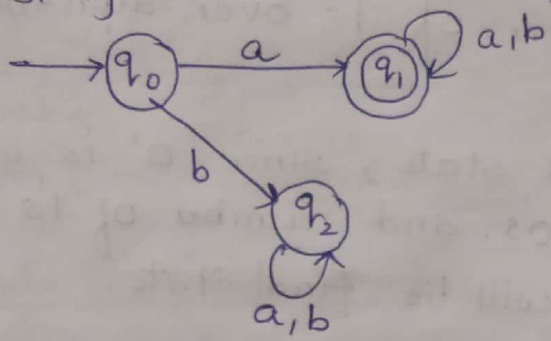
$= \delta(q_1,1)$

$= q_0 \in F$

$\therefore$ String 110101 is accepted.

Scanned with CamScanner

2) Design a DFA for language L over {a, b} that contains set of strings that begin with a.. and check whether string abbab is accepted or not.



$$\delta^*(q_0, abbab) = \delta(\delta^*(q_0, abba)b)$$
$$= \delta(\delta(\delta^*(q_0, abb)a)b)$$
$$= \delta(\delta(\delta(\delta^*(q_0, ab)b)a)b)$$
$$= \delta(\delta(\delta(\delta(\delta(q_0, a)b)b)a)b$$
$$= \delta(\delta(\delta(\delta(q_1, b)b)a)b$$
$$= \delta(\delta(\delta(q_1, b)a)b)$$
$$= \delta(\delta(q_1, a)b)$$
$$= \delta(q_1, b)$$
$$= q_1 \in F$$

Since q_1 is final state so string abbab is accepted by finite Automata.

Scanned with CamScanner

Non-Deterministic finite Automata

A Non-Deterministic finite automata (NFA) has the Power to be in several states at once.

Let M be NFA then M is defined as 5-tuple

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

Q : Is a non-empty finite set of states in NFA.

Σ : Is a non-empty finite set of input symbols or alphabet

q_0 : Is a starting state / An initial state $q_0 \in Q$

F : Is a non-empty set of final state or accepting states $F \subseteq Q$.

δ : Is a function called a transition function that takes two arguments a state and a input symbol, it returns a single state. $\delta : Q \times \Sigma \longrightarrow Q$

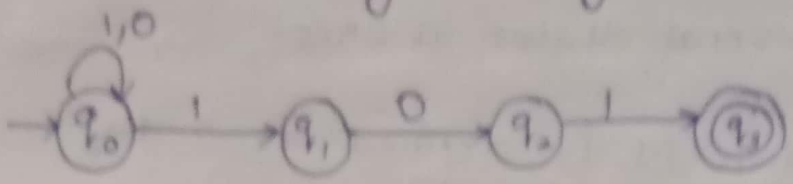
$$\delta(q_0, a) = q_1$$

features

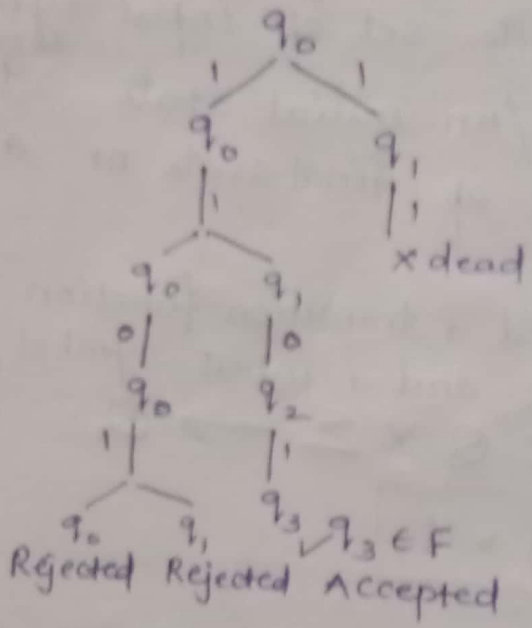
1) A state may have no transition on particular symbol.
2) Ability to transition to move more than one state on a given input symbol.
3) Ability to take a move without reading any input symbol.

Scanned with CamScanner

1) construct a NFA for binary language that accepts all the string ending with 101.



Construct a computation Tree for the string 1101 for the above NDFA.

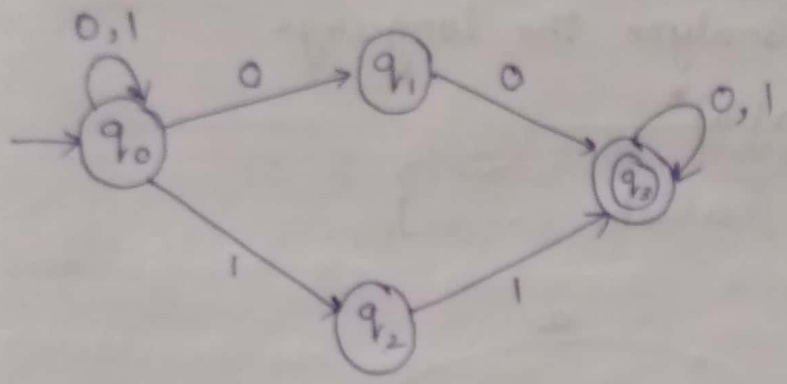


Computation Tree

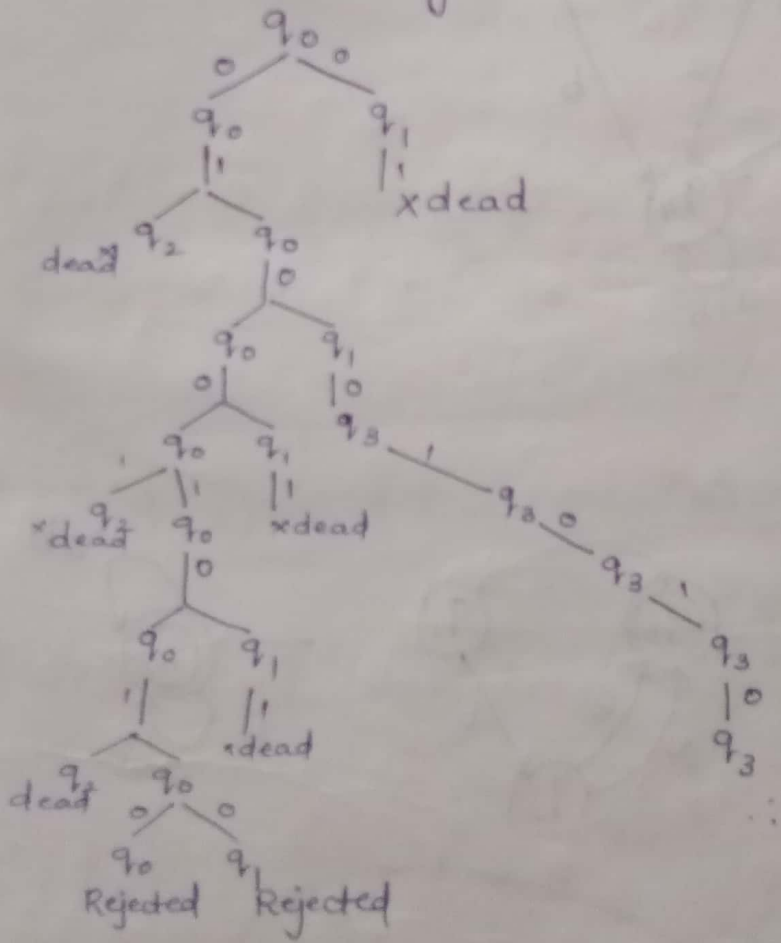
The branch or guess transition can be represented as a tree going downwards with the children of a node its possible successor.

Scanned with CamScanner

2) Design a NFA for binary string that contain either '00' or '11'



Construct a computation Tree for the string 01001010 & check whether string is accepted or not.



∴ string is accepted.

Scanned with CamScanner

3) Design a DFA & NFA for the language

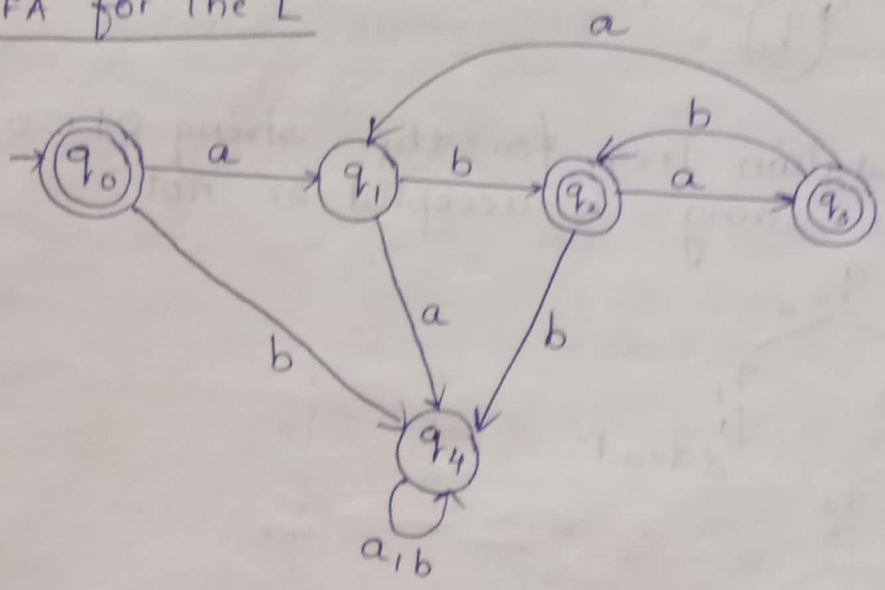
$L = (ab \cup aba)^*$.

Soln Let us first analyze the language

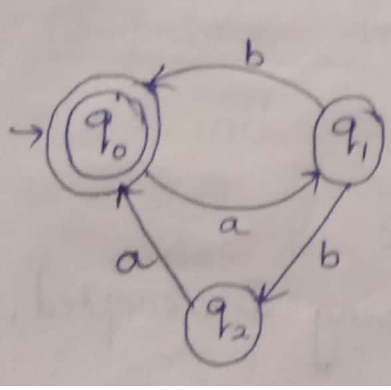
$L = (ab \cup aba)^*$

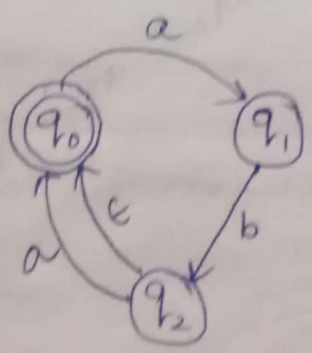
$L = \{ab, aba, abab, \ldots\}$

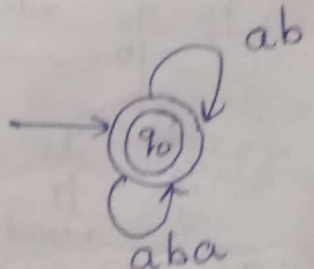
DFA for the L



NFA for the L







Scanned with CamScanner

Transition function in NFA

1) For each state there can be 0, 1, 2 or more transition corresponding to an input symbol.

2) **Branch / Guess transition**

If NFA gets to a state with more than one possible transition for the input symbol, it is define as guess/ branch transition

3) **Dead Transition**

If NFA gets to a state where there is no valid transition for the next input symbol then guess or branch transition dies.

Transition function is

$$\delta : Q \times \Sigma \longrightarrow K \qquad K \subseteq Q$$

Extended Transition function for NFA

$$\delta^*(q, x) = \{P_1, P_2 \ldots P_k\}$$

Let

$$\bigcup_{i=1}^{k} \delta(P_i, a) = \{r_1, r_2, r_3 \ldots r_m\}$$

Then

$$\delta^*(q, w) = \{r_1, r_2 \ldots r_m\}$$

Language of NFA

Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA the function $\delta^* : Q \times \Sigma = 2^Q$ is defined as

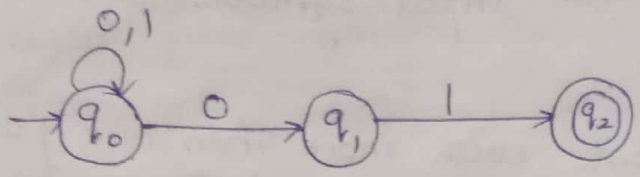
i) for any $q \in Q$, $\delta^*(q, \epsilon) = \{q\}$

ii) for any $q \in Q$, $y \in \Sigma^*$ and $a \in \Sigma$

$L(M) = \{\delta^*(q, ya) \cap F \neq \phi\}$ is the language of M i.e NFA. $L(M)$ is the contains strings over $\Sigma = \{y, a\}$ such that $\delta(q, ya)$ contains atleast one accepting state.

Scanned with CamScanner

Q) Consider NFA and describe the processing of string 00101 by it.



Soln It is a NFA accepting all strings end in 01.

Let us use $\hat{\delta}$ or δ^* to describe the processing of input 00101 by NFA.

$$\hat{\delta}(q_0, \epsilon) = q_0$$

$$\hat{\delta}(q_0, 0) = \delta(q_0, 0) = \{q_0, q_1\}$$

$$\hat{\delta}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \phi = \{q_0, q_1\}$$

$$\hat{\delta}(q_0, 001) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$$

$$\hat{\delta}(q_0, 0010) = \delta(q_0, 0) \cup \delta(q_2, 0) = \{q_0, q_1\} \cup \phi = \{q_0, q_1\}$$

$$\hat{\delta}(q_0, 00101) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$$

Language Accepted by DFA

- A string x is accepted by a DFA $D = \{Q, \Sigma, \delta, q_s, F\}$ only if extended transition $\delta^*(q_s, x) = P$, $P \in F$.
- The language accepted by DFA is denoted by $L(D)$ where
$$L(D) = \{x \mid \delta^*(q_s, x) = P \text{ , } P \in F\}$$
- If L is any language over Σ, L is accepted or recognized by D iff $L = L(D)$
- A language L over Σ is regular iff there is a DFA with input alphabet Σ that accepts L.

Scanned with CamScanner

Language Accepted by NFA

Let $N = \{Q, \Sigma, \delta, q_s, F\}$ be a NFA, the string $x \in \Sigma^*$ is accepted by N if

$$\delta^*(q_s, x) \cap F \neq \phi$$

The language accepted by NFA $L(N)$ where

$$L(N) = \{\omega \mid \delta^*(q_s, \omega) \cap F \neq \phi\}$$

i,e $L(N)$ is a set of string over Σ such that extended transition $\delta^*(q_s, \omega)$ has atleast one accepting state.

Scanned with CamScanner

Reduction of the Number of states in Finite Automata

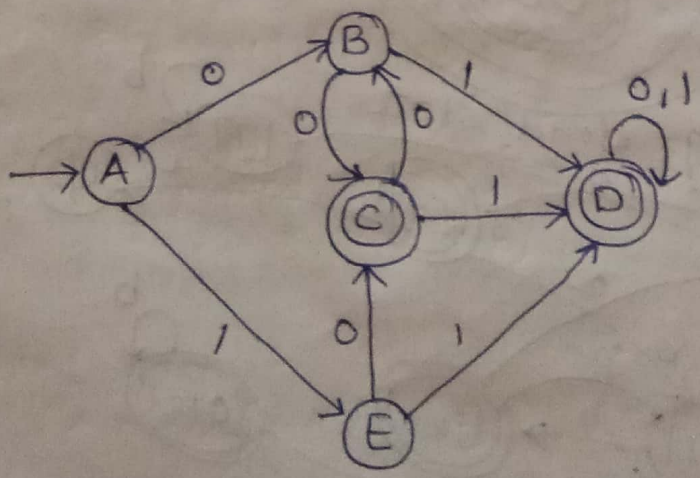
DFA Minimization using Myhill-Nerode Theorem

Input : DFA
Output : Minimized DFA.

Step1 :- Draw a table for all pairs of states (P,Q)
Step2 : Mark all pairs where $P \in F$ and $Q \notin F$.
Step3 : If there are any unmarked pairs (P,Q) such that $[\delta(P,X), \delta(Q,X)]$ is marked. then mark [P,Q], where x is input symbol.
Repeat this until no more markings can be made.
step4 : combine all the unmarked pairs and then make them a single state in minimized DFA.

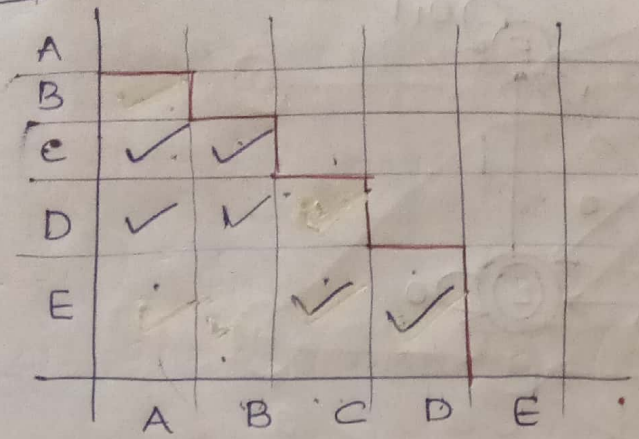
Example.

1) Minimize the given DFA using Myhill - Nerode Theorem.

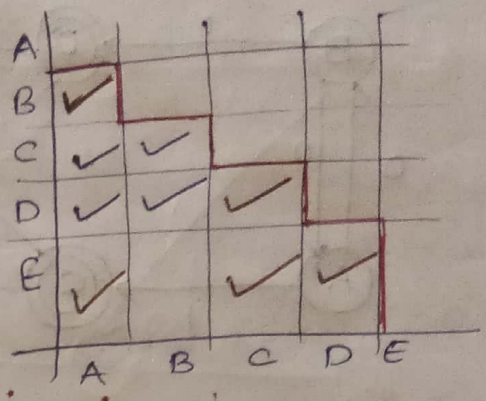


Scanned with CamScanner

Step 2:



Step 3



Step 3: (Transitions)

(A,B): $\delta(A,0) = B$ (B,C) marked
$\delta(B,0) = C$
∴ Mark Pair (A,B)

$\delta(A,1) = E$ (E,D)
$\delta(B,1) = D$ ∴ marked

(D,C): $\delta(D,0) = D$ (D,B) marked
$\delta(C,0) = B$
∴ Mark pair (D,C)

$\delta(D,1) = D$ (D,D) not present
$\delta(C,1) = D$

(E,A): $\delta(E,0) = C$ (C,B) marked
$\delta(A,0) = B$
∴ Mark Pair (E,A)

$\delta(E,1) = D$ (D,E) ∴ Marked
$\delta(A,1) = E$

(E,B) = $\delta(E,0) = C$ (C,C) not present
$\delta(B,0) = C$

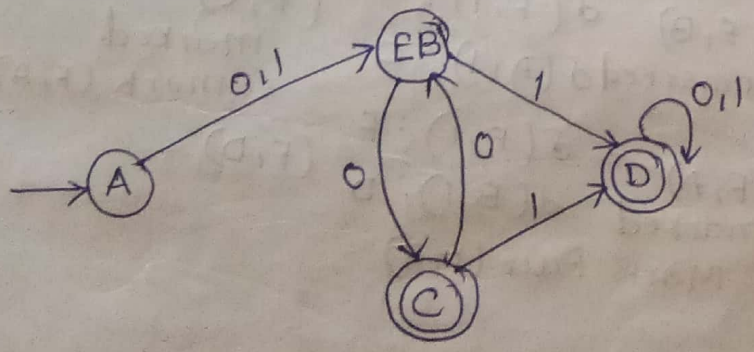
$\delta(E,1) = D$ (D,D) (not present)
$\delta(B,1) = D$

Step 4:

Combine all unmarked pairs and then make them a single state.

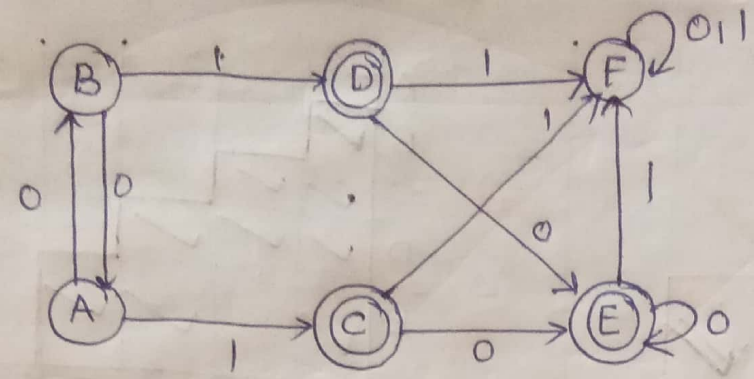
unmarked pair = (E, B)

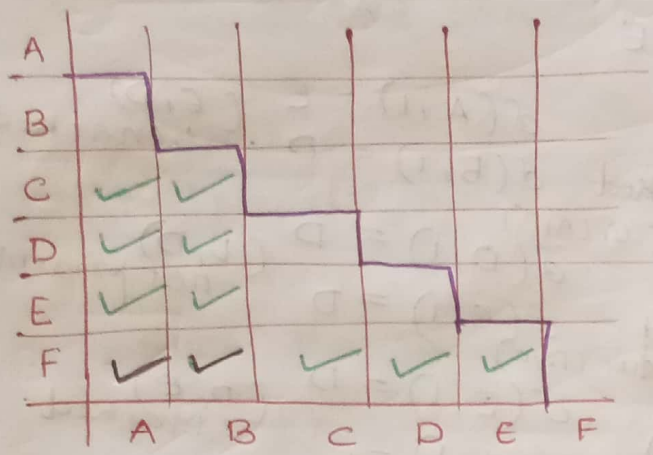
The minimized DFA is



Scanned with CamScanner

2)





Step 3

$(B,A) \rightarrow \delta(B,0):A \quad \delta(A,0):B$ — (A,B) unmarked

$\delta(B,1):D \quad \delta(A,1):C$ — (D,C) unmarked

$(D,C) \rightarrow \delta(D,0):E \quad \delta(C,0):E$ — (E,E) no pair

$\delta(D,1):F \quad \delta(C,1):F$ — (F,F) no pair - ignore.

$(E,C) \rightarrow \delta(E,0):E \quad \delta(C,0):E$ — (E,E) not present

$\delta(E,1):F \quad \delta(E,1):F$ — (F,F) not present

$(E,D) \rightarrow \delta(E,0):E \quad \delta(D,0):E$ — (E,E) not present

$\delta(E,1):F \quad \delta(D,1):F$ — (F,F) not present

$(F,A) \rightarrow \delta(F,0):F \quad \delta(A,0):B$ — (F,B) unmarked

$\delta(F,1):F \quad \delta(A,1):C$ — (F,C) marked

∴ mark (F,A) pair

$(F,B) \rightarrow \delta(F,0):F \quad \delta(B,0):A$ — (F,A) marked

$\delta(F,1):F \quad \delta(B,1):D$ — (F,D)

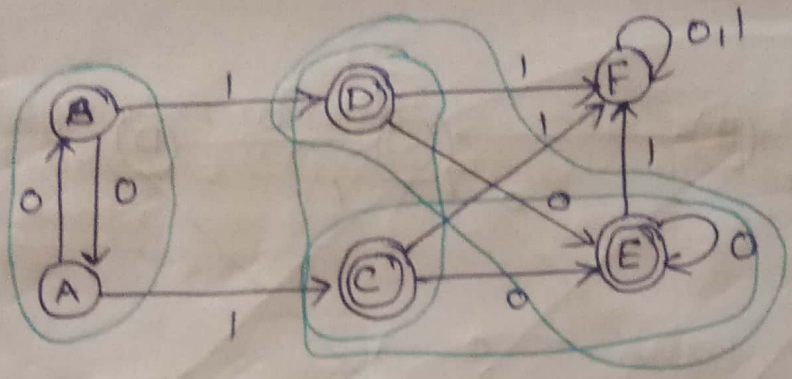
∴ Mark Pair (F,B)

Step 4:-

Unmarked pairs :-

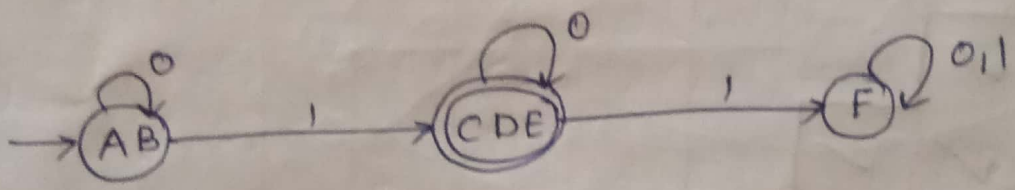
(A,B), (D,C), (E,C), (E,D)

Scanned with CamScanner



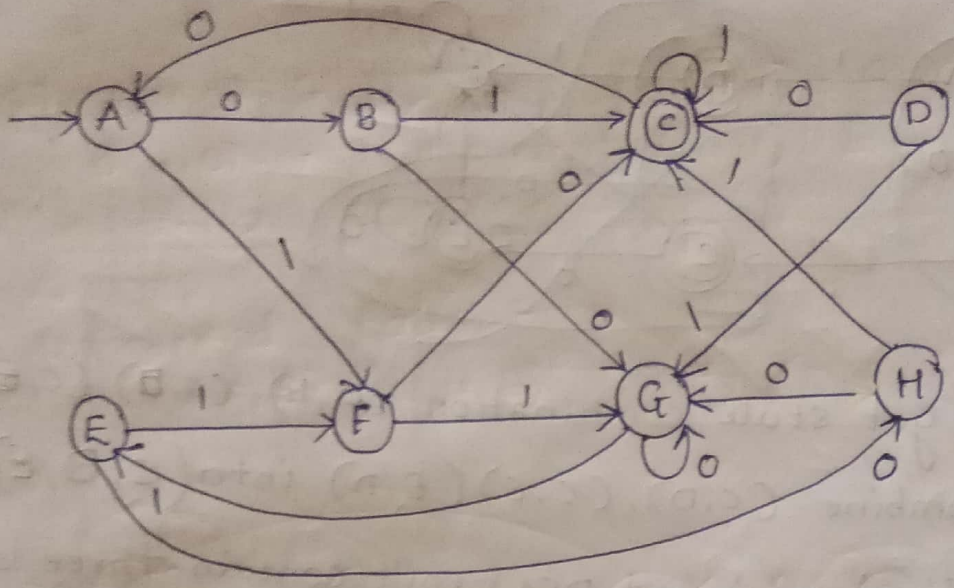
we have got state combination (A,B), (C,D) (C,E), (E,D)
we can combine (C,D), (C,E) (E,D) into (C, D, E)
So the final minimized DFA will contain three states
AB, CDE, and F..

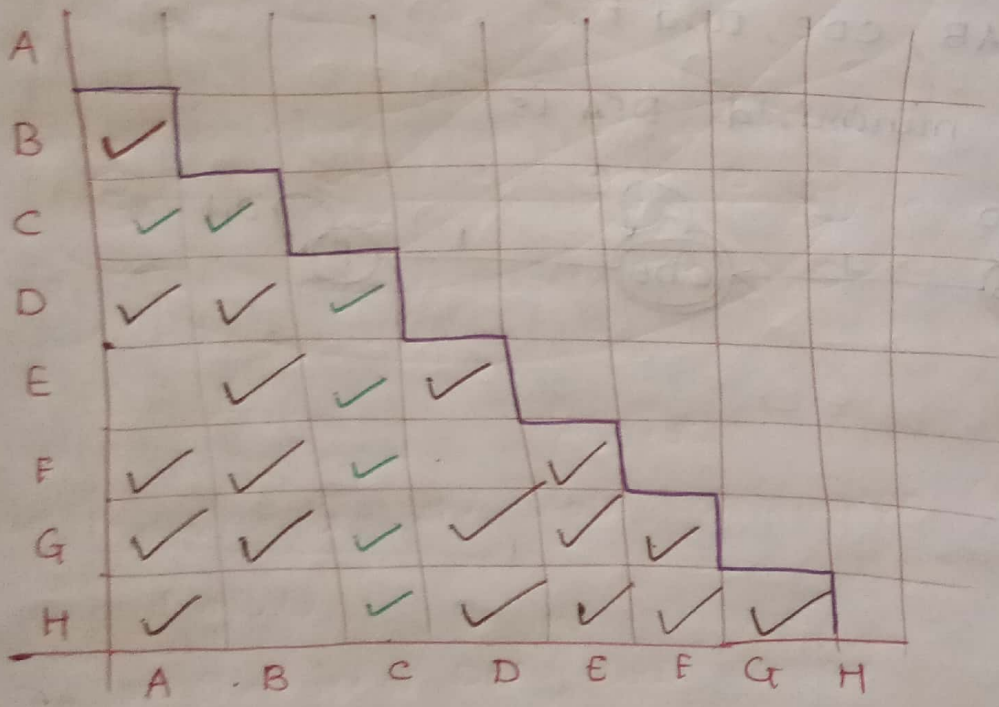
∴ The minimized DFA is



Scanned with CamScanner

3)





Step 3:-

(A,B) → $\delta(A,0)$: B, $\delta(B,0)$: G → (B,G) unmarked

$\delta(A,1)$: F, $\delta(B,1)$: C → (F,C) marked ∴ Mark pair (A,B)

(D,A) → $\delta(D,0)$: C, $\delta(A,0)$: B → (C,B) marked ∴ Mark pair (D,A)

$\delta(D,1)$: G, $\delta(A,1)$: F → (G,F) unmarked.

(D,B) → $\delta(D,0)$: C, $\delta(B,0)$: G → (C,G) Marked ∴ Mark pair (D,B)

$\delta(D,1)$: G, $\delta(B,1)$: C → (G,C) Marked ∴ Mark pair (D,B)

(E,A) $\delta(E,0)$: H, $\delta(A,0)$: B → (H,B) unmarked

$\delta(E,1)$: F, $\delta(A,1)$: F → (F,F) not present

(E,B) $\delta(E,0)$: H, $\delta(B,0)$: G → (H,G) unmarked

$\delta(E,1)$: F, $\delta(B,1)$: C → (F,C) Marked ∴ Mark pair (E,B)

(E,D) $\delta(E,0)$: H, $\delta(D,0)$: C → (H,C) Marked ∴ Mark pair (E,D)

$\delta(E,1)$: F, $\delta(D,1)$: G → (F,G) unmarked

Scanned with CamScanner

(F,A) → δ(F,0) : C (C,B) Marked
δ(A,0) : B
∴ Mark Pair (F,A)

δ(F,1) : G (G,F) unmarked
δ(A,1) : F

(F,B) → δ(F,0) : C (C,G) Marked
δ(B,0) : G
∴ Mark Pair (F,B)

δ(F,1) : G (G,C) Marked
δ(B,1) : C
∴ Mark pair (F,B)

(F,D) → δ(F,0) : C (C,C) not present
δ(D,0) : C

δ(F,1) : G (G,G) not present
δ(D,1) : G

(F,E) → δ(F,0) : C (C,H) Marked
δ(E,0) : H
∴ Mark Pair (F,E)

δ(F,1) : G (G,F) unmarked
δ(E,1) : F

(G,A) → δ(G,0) : G (G,B) unmarked
δ(A,0) : B

δ(G,1) : E (E,F) Marked
δ(A,1) : F
∴ Mark Pair (G,A)

(G,B) δ(G,0) : G (G,G) not present
δ(B,0) : G

δ(G,1) : E (E,C) Marked
δ(B,1) : C
∴ Mark pair (G,B)

(G,D) δ(G,0) : G (G,C) Marked
δ(D,0) : C
∴ Markpair (G,D)

δ(G,1) : E (E,G) unmarked
δ(D,1) : G

(G,E) δ(G,0) : G (G,H) unmarked
δ(E,0) : H

δ(G,1) : E (E,F) Marked
δ(E,1) : F
∴ Mark pair (G,E)

(G,F) δ(G,0) : G (G,C) Marked
δ(F,0) : C
∴ Mark pair (G,F)

δ(G,1) : E (E,G) unmarked
δ(F,1) : G

(H,A) δ(H,0) : G (G,B) Marked
δ(A,0) : B
∴ Mark Pair (H,A)

δ(H,1) : C (C,F) Marked
δ(A,1) : F
∴ Mark Pair (H,A)

(H,B) δ(H,0) : G (G,G) not present
δ(B,0) : G

δ(H,1) : C (C,C) not present
δ(B,1) : C

(H,D) δ(H,0) : G (G,C) Marked
δ(D,0) : C
∴ Mark Pair (H,D)

δ(H,1) : C (C,G) Marked
δ(D,1) : G
∴ Mark Pair (H,D)

(H,E) δ(H,0) : G (G,H) unmarked
δ(E,0) : H

δ(H,1) : C (C,F) Marked
δ(E,1) : F
∴ Mark pair (H,E)

(H,F) δ(H,0) : G (G,C) Marked
δ(F,0) : C
∴ Mark Pair (H,F)

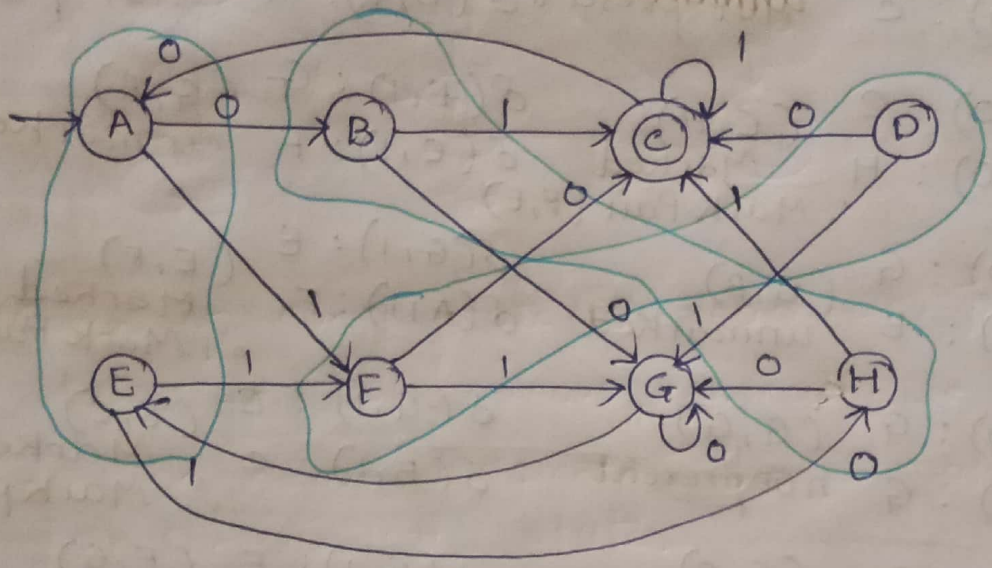
δ(H,1) : C (C,G) Marked
δ(F,1) : G
∴ Mark Pair (H,F)

Scanned with CamScanner

(H, G) $\delta(H,0)$: G (G,G) $\delta(H,1)$: C (C, E)
$\delta(G,0)$: G not present $\delta(G,1)$: E Marked
∴ Mark Pair (H, G)

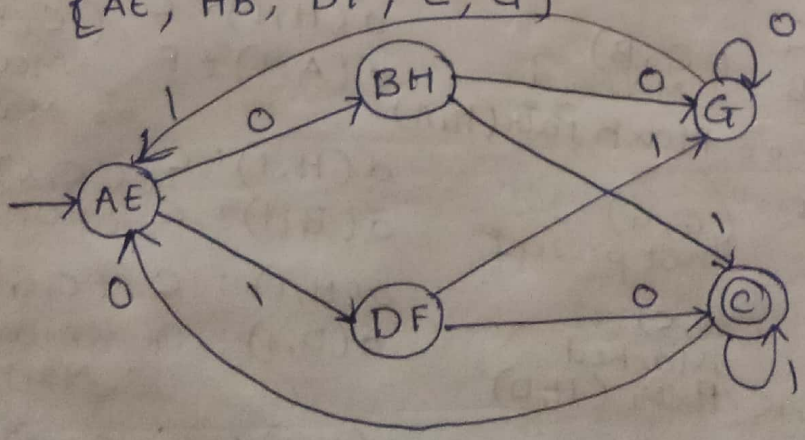
Step 4

∴
Unmarked Pairs :-
(A, E) , (H, B) , (D, F)



we have got the state combination (A, E), (H, B) and (D, F)

∴ so the Final DFA will contain
{AE, HB, DF, C, G}



Scanned with CamScanner

DFA Minimization Using Equivalence Theorem

If X and Y are two states in a DFA, we can combine these two states into {X, Y}. If they are not distinguishable. Two states are distinguishable, if there is at least one string S, such that one of $\delta(X, S)$ and $\delta(Y, S)$ is accepting and another is not accepting. Hence, A DFA is minimal if and only if all states are distinguishable.

Algorithm

Step1 : All these states Q are divided in two partitions: final states and non-final states and is denoted by P_0. All these states in Partitions are 0^{th} equivalent. Take a counter K and initialize it with 0.

Step2 : Increment K by 1. For each partition in P_K, divide the states in P_2 into two Partitions if they are K-distinguishable. Two states within this Partition X and Y are K-distinguishable if there is an input S so that $\delta(X, S)$ and $\delta(Y, S)$ are (K-1) distinguishable.

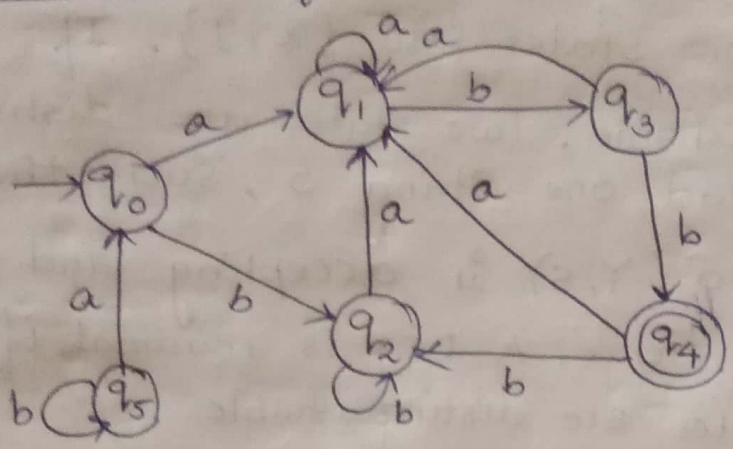
Step3 : If $P_K \neq P_{K-1}$, then Repeat step2 otherwise Go to Step4.

Step4 : Combine K^{th} equivalent sets and make them the new states of the reduced DFA.

Scanned with CamScanner

Let us consider the following example of DFA.

Minimization of DFA using equivalence Theorem



Check if every state is reachable from initial state. If not, remove unreachable state.

In the above Automata, q_5 is not reachable from the initial state.

∴ We remove state q_5.

Transition Table

States	a	b
→ q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_1	q_2
q_3	q_1	* q_4
* q_4	q_1	q_2

Let us apply algorithm to this DFA.

0-equivalence state - find the final and non-final state.

$[q_0\ q_1\ q_2\ q_3]$, $[q_4]$

Scanned with CamScanner

1 - equivalence

$= [q_0, q_1, q_2]\ [q_3]\ [q_4]$

2 - equivalence

$= [q_0, q_2]\ [q_1]\ [q_3]\ [q_4]$

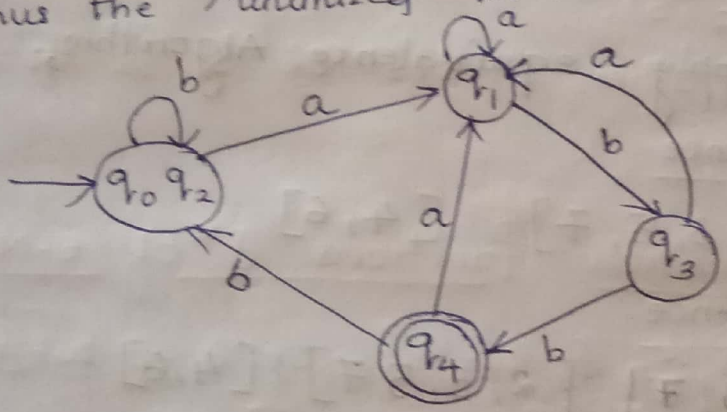
3 - equivalence

$= [q_0\ q_2]\ [q_1]\ [q_3]\ [q_4]$

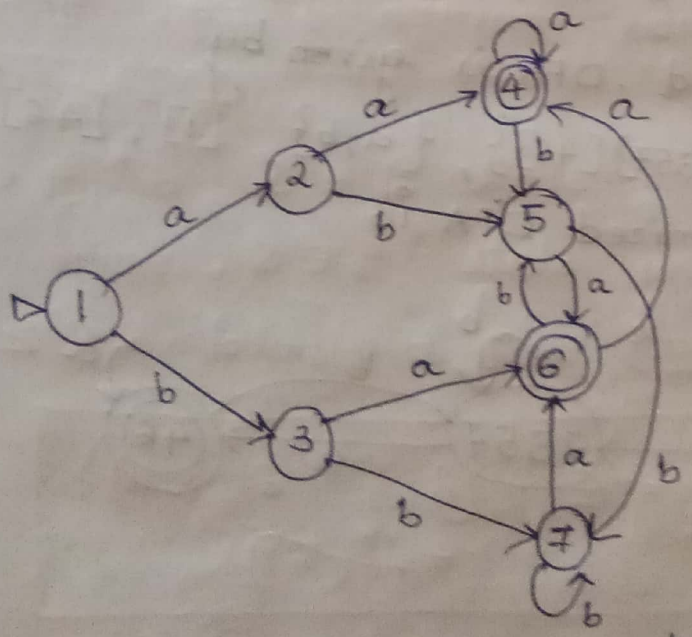
Thus the Minimized DFA is given by

$M' = (\{[q_0 q_2][q_1][q_3][q_4], \{a, b\}, 1, [q_4]\})$

Thus the Minimized DFA is



a)



Check if there are ~~every~~ state reachable from initial state

∵ from initial state we reach to every other state, no need to remove any state.

Scanned with CamScanner

Transition Table

State	a	b
→1	2	3
2	4	5
3	6	7
* 4	4	5
5	6	7
* 6	4	5
7	6	7

Let us apply equivalence Algorithm,

0-equivalence

[1, 2, 3, 5, 7] [4, 6]

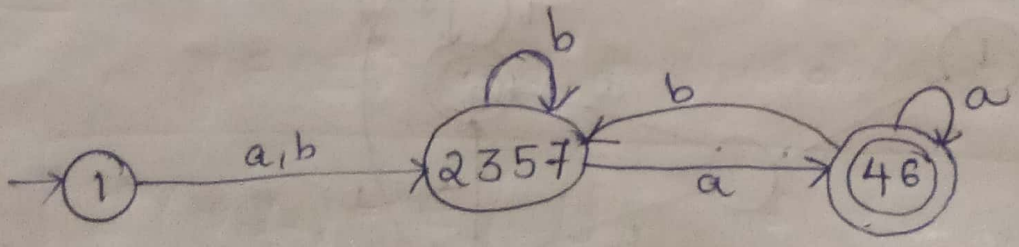
1-equivalence

[1] , [2, 3, 5 7] [4, 6]

The Minimized DFA is given by

$M' = (\{[1], [2357], [46]\}, \{a, b\}, [1], [46]\}$

∴ The minimized DFA is



Scanned with CamScanner

DFA Minimization Using Equivalence Theorem

If X and Y are two states in a DFA, we can combine these two states into {x, y}. If they are not distinguishable. Two states are distinguishable, if there is at least one string S, such that one of $\delta(X, S)$ and $\delta(Y, S)$ is accepting and another is not accepting. Hence, A DFA is minimal if and only if all states are distinguishable.

Algorithm

Step 1 : All these states Q are divided in two partitions: final states and non-final states and is denoted by P_0. All these states in Partitions are 0^{th} equivalent. Take a counter K and initialize it with 0.

Step 2 : Increment K by 1. For each partition in P_K, divide the states in P_2 into two Partitions if they are K-distinguishable. Two states within this Partition X and Y are K-distinguishable if there is an input S so that $\delta(X, S)$ and $\delta(Y, S)$ are (K-1) distinguishable.

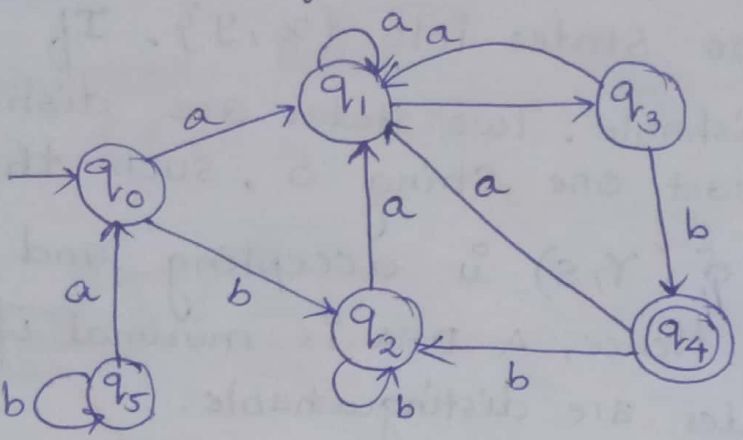
Step 3 : If $P_K \neq P_{K-1}$, then Repeat step 2 otherwise Go to Step 4.

Step 4 : Combine K^{th} equivalent sets and make them the new states of the reduced DFA.

Scanned with CamScanner

Let us consider the following example of DFA.

Minimization of DFA using equivalence Theorem



Check if every state is reachable from initial state. If not, remove unreachable state.

In the above Automata, q_5 is not reachable from the initial state.

∴ We remove state q_5.

Transition Table

States	a	b
→ q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_1	q_2
q_3	q_1	* q_4
* q_4	q_1	q_2

Let us apply algorithm to this DFA.

0-equivalence state - find the final and non-final state.

$[q_0\ q_1\ q_2\ q_3]$, $[q_4]$

Scanned with CamScanner

1-equivalence

$= [q_0, q_1, q_2]\ [q_3][q_4]$

2-equivalence

$= [q_0, q_2]\ [q_1]\ [q_3]\ [q_4]$

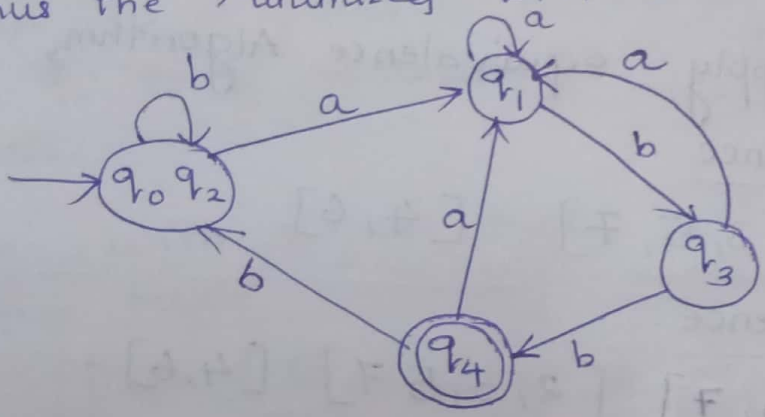
3-equivalence

$= [q_0\ q_2]\ [q_1][q_3][q_4]$

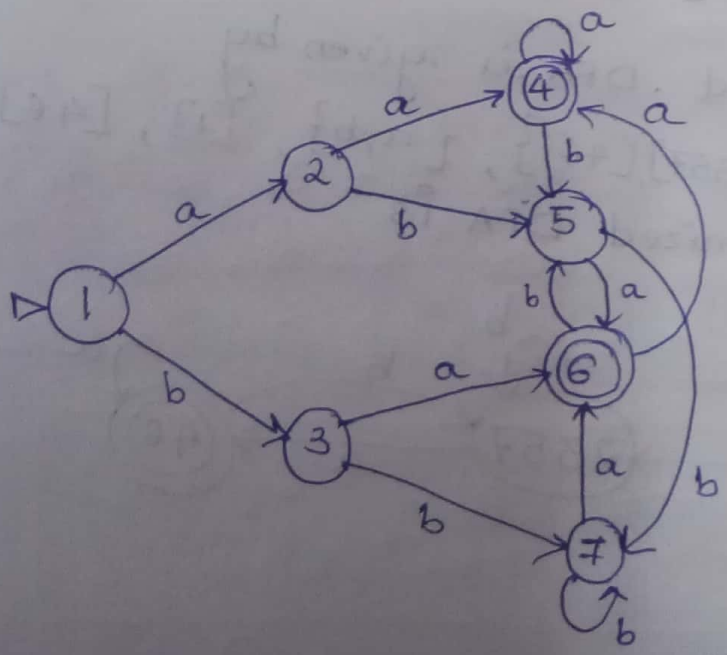
Thus the Minimized DFA is given by

$M' = (\{[q_0 q_2][q_1][q_3][q_4], \{a,b\}, 1, [q_4]\})$

Thus the Minimized DFA is



2)



Check if there are ~~every~~ state reachable from initial state

∵ from initial state we reach to every other state, no need to remove any state.

Scanned with CamScanner

Transition Table

State	a	b
→1	2	3
2	4	5
3	6	7
*4	4	5
5	6	7
*6	4	5
7	6	7

Let us apply equivalence Algorithm,

0-equivalence

[1, 2, 3, 5, 7] [4, 6]

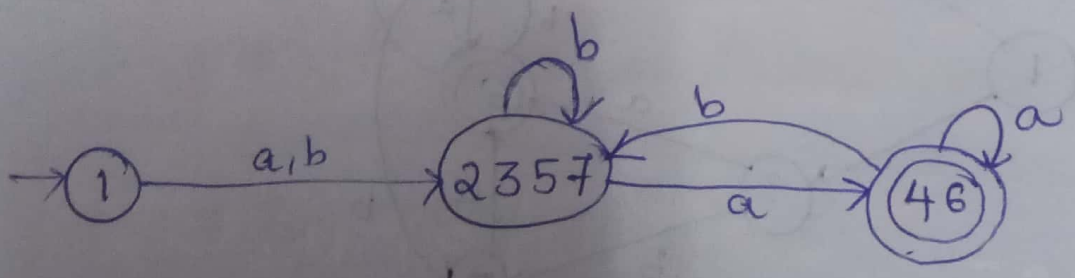
1-equivalence

[1], [2, 3, 5 7] [4, 6]

The Minimized DFA is given by

$M' = (\{[1], [2357], [46]\}, \{a, b\}, [1], [46]\}$

∴ The minimized DFA is



Scanned with CamScanner

NFA with ϵ-Transitions

The Transitions made without symbols are called as ϵ-transitions.

Consider the following NFA

The Finite Automata is a NFA with ϵ-transitions because it is possible to make from State q_0 to q_1 without consuming any of input symbols.

NFA with ϵ-transitions will also be denoted as five tuple.

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

where Q, Σ, q_0 and F are having usual Meaning and transition function δ defines a mapping from

$$Q \times (\Sigma \cup \epsilon) \text{ to } 2^Q.$$

Scanned with CamScanner

ϵ - closure

Let $M = \{Q, \Sigma, \delta, q_0, F\}$ be a ϵ-NFA, and let S be a set, $S \subseteq Q$.

The ϵ-closure of S is the set $E(S)$ is defined by

Rule 1 :- Every state $q \in S$, is in $\epsilon(S)$.

Rule 2 :- for any state $q \in S$, the $\delta(q, \epsilon)$ is in $E(S)$.

Rule 3 :- No other states of Q is in $\epsilon(S)$.

Extended Transition functions δ^* of ϵ-NFA

Let $M = \{Q, \Sigma, \delta, q_0, F\}$ be a ϵ-NFA, the extended transition function δ^* can be defined as.

Rule 1 :- for any state $q \in Q$, $\delta^*(q, \epsilon) = \epsilon(q$

Rule 2 :- for any state $q \in Q$, some string $y \in \Sigma^*$ and a symbol $a \in \Sigma$.

$$\delta^*(q, ya) = \epsilon\left(\bigcup_{r \in \delta^*(q,y)} \delta(r, a)\right)$$

Scanned with CamScanner

Equivalence of ϵ-NFA and NFA

Theorem :- If $L \subseteq \Sigma^*$ is a language accepted by a ϵ-NFA $M = (Q, \Sigma, \delta, q_0, F)$, then there is a NFA $M' = (Q', \Sigma, \delta', q_0, F')$ without ϵ-transitions that also accepts same language L.

Proof :-

The NFA without ϵ-transitions M' can be defined as.

$$Q' = Q$$

$$q_0 = q_0$$

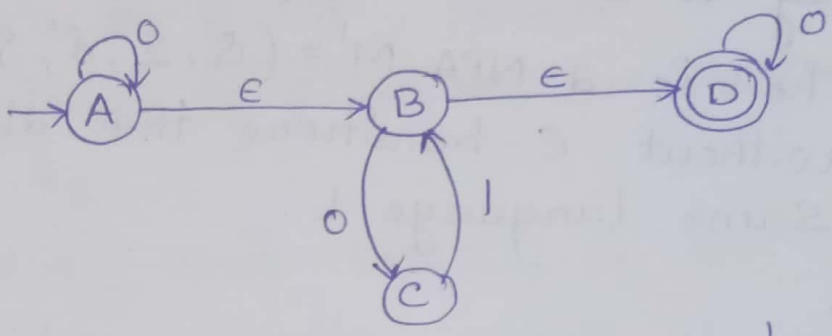
$$F' = \begin{cases} F \cup q_0 ; & \epsilon(q_0) \cap F \neq \phi \\ F ; & \text{otherwise} \end{cases}$$

δ^* can be defined as for any state $q \in Q'$ and $a \in \Sigma$,

$$\delta'(q, a) = \delta^*(q, \epsilon a)$$

Scanned with CamScanner

1) Obtain the equivalent NFA from given ϵ-NFA.



The NFA without ϵ-transitions M' can be defined as

$Q = \{A, B, C, D\}$

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

$q_0 = \{A\}$

$F = \{D\}$

$\epsilon(A) = \{A, B, D\} \cap \{D\} = D \neq \phi$

$F' = \{A, D\}$ as $\epsilon(A) \cap F \neq \phi$

$$\delta'(A,0) = \delta^*(A,0) = \delta(\epsilon(A), 0)$$
$$= \delta(\{A,B,D\}, 0)$$
$$= \delta(A,0) \cup \delta(B,0) \cup \delta(D,0)$$
$$= \{A\} \cup \{C\} \cup \{D\}$$
$$= \epsilon(\{A, C, D\})$$
$$= \{A, C, D, B\}$$

$$\delta'(A,1) = \delta^*(A,1) = \delta(\epsilon(A), 1)$$
$$= \delta(\{A,B,D\}, 1)$$
$$= \delta(A,1) \cup \delta(B,1) \cup \delta(D,1)$$
$$= \phi \cup \phi \cup \phi$$
$$= \phi$$

Scanned with CamScanner

$\delta'(B,0) = \delta^*(B,0) = \delta(\epsilon(B), 0)$

$= \delta(\{B,D\}, 0)$

$= \delta(B,0) \cup \delta(D,0)$

$= \{C\} \cup \{D\}$

$= \{C,D\}$

$\delta'(B,1) = \delta^*(B,1) = \delta(\epsilon(B), 1)$

$= \delta(\{B,D\}, 1)$

$= \delta(B,1) \cup \delta(D,1)$

$= \phi \cup \phi$

$= \phi$

$\delta'(C,0) = \delta^*(C,0) = \delta(\epsilon(C), 0)$

$= \delta(\{C\}, 0)$

$= \delta(C,0)$

$= \phi$

$\delta'(C,1) = \delta^*(C,1) = \delta(\epsilon(C), 1)$

$= \delta(\{C\}, 1)$

$= \delta(C,1)$

$= B \quad \because \epsilon(B) = \{B,D\}$

$\delta'(D,0) = \delta^*(D,0) = \delta(\epsilon(D), 0)$

$= \delta(\{D\}, 0)$

$= D \quad \because \epsilon(D) = \{D\}$

$\delta'(D,1) = \delta^*(D,1) = \delta(\epsilon(D), 1)$

$= \delta(D,1)$

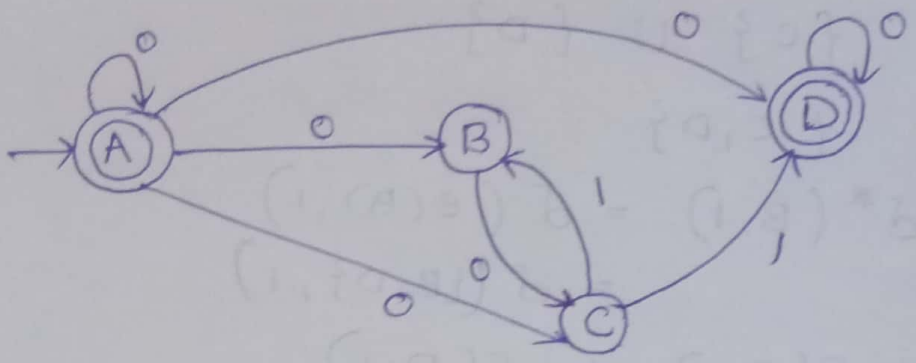
$= \phi$

Transition Table

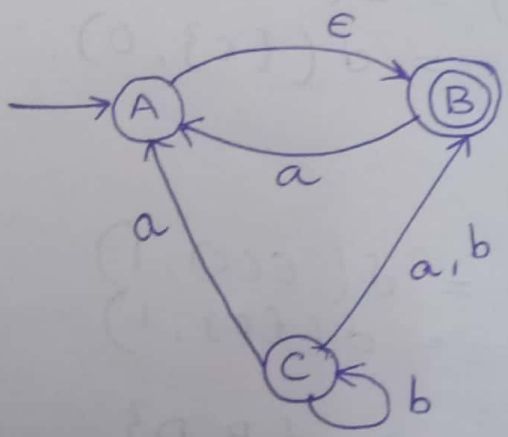
State	0	1
→ A	{A, C, D, B}	ϕ
B	{C, D}	ϕ
C	ϕ	{B, D}
* D	{D}	ϕ

Scanned with CamScanner

Resultant NFA without ϵ-transitions



2) Obtain the equivalent NFA from ϵ-NFA.



The NFA without ϵ-Moves $M = (Q, \Sigma, \delta', q_0, F')$ is defined as

$Q = \{A, B, C\}$

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

$q_0 = \{A\}$

$F' = \{A, B\}$ $\because \epsilon(A) = \{A, B\}$

$\therefore F' = \begin{cases} F \cup [q_0] & ; \text{ if } \epsilon(q_0) \text{ contains any final state in } F. \\ F & ; \text{ Otherwise} \end{cases}$

Scanned with CamScanner

$$\delta'(A,a) = \delta^*(A,a) = \delta(\epsilon(A), a)$$
$$= \delta(\{A,B\}, a)$$
$$= \delta(A,a) \cup \delta(B,a)$$
$$= \phi \cup \phi = \phi$$

$$\delta'(A,b) = \delta^*(A,b) = \delta^*(\epsilon(A), b)$$
$$= \delta^*(\{A,B\}, b)$$
$$= \delta(A,b) \cup \delta(B,b)$$
$$= \phi \cup \phi = \phi$$

$$\delta'(B,b) = \delta^*(B,b) = \delta^*(\epsilon(B), b)$$
$$= \delta(B,b)$$
$$= \phi$$

$$\delta'(B,a) = \delta^*(B,a) = \delta^*(\epsilon(B), a)$$
$$= \delta(B,a)$$
$$= \{A\} \quad \because \epsilon(A) = \{A,B\}$$

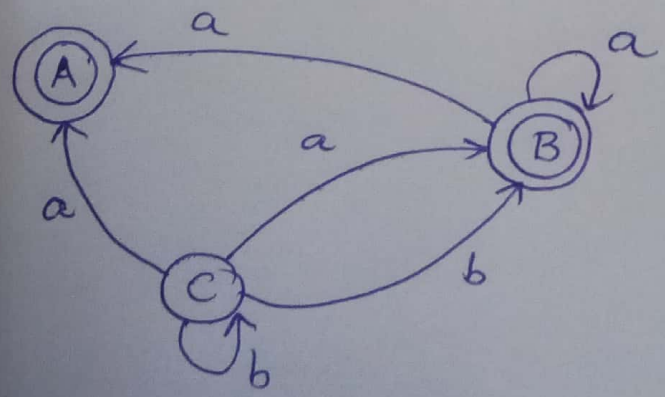
$$\delta'(C,a) = \delta^*(C,a) = \delta^*(\epsilon(C), a)$$
$$= \delta(\{C\}, a)$$
$$= \{A\} \quad \because \epsilon(A) = \{A,B\}$$

$$\delta'(C,b) = \delta^*(C,b) = \delta^*(\epsilon(C), b)$$
$$= \delta(\{C\}, b\} = \{C\}$$
$$= \{C,B\}$$

Transition Table.

state	a	b
→ A	ϕ	ϕ
B	{A,B}	ϕ
C	{A,B}	{C,B}

Resultant NFA without ϵ-Moves



Scanned with CamScanner

Algorithm

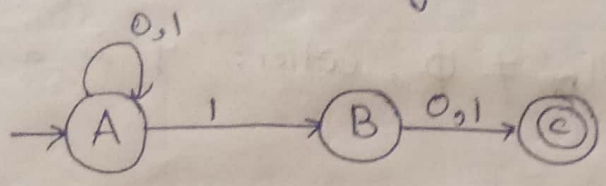
Input :- A NDFA
output :- An equivalent DFA

Steps

Step 1 :- create state table from the given DFA
Step 2 :- create a blank state table under possible input alphabets for the equivalent DFA.
Step 3 :- Mark the state of the DFA by q_0 (same as NDFA)
Step 4 :- Find out the combination of states $\{Q_0, Q_1, \cdots Q_n\}$ for each possible input alphabet.
Step 5 :- Each time we generate a new DFA state under the input alphabet columns, we have to apply step 4 again, otherwise go to step 6.
Step 6 :- The states which contain any of the final states of the NDFA is the final states of the equivalent DFA.

Scanned with CamScanner

1) Convert the following NFA for its equivalent DFA.



$N = \{ \{A, B, C\}, \{0, 1\}, \delta_N, A, \{C\} \}$

The δ is defined as :

$\delta_N(A,0) = A \quad \delta_N(A,1) = \{A, B\}$

$\delta_N(B,0) = C \quad \delta_N(B,1) = C$

$\delta_N(C,0) = \phi \quad \delta_N(C,1) = \phi$

The DFA D can be defined as

$D = \{ Q_D, \Sigma, \delta_D, q_D, F_D \}$

$Q_D = \{ [A], [B], [C], [AB], [BC], [AC], [ABC], \phi \}$

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

$q_D = \{A\}$

$F_D = \{[C], [BC], [AC], [ABC]\}$

By subset construction Method

$Q_N = 3$

$Q_D = 2^{Q_N} = 2^3 = 8$ states.

$\delta_D([A], 0) = \delta_N(A,0) = [A]$

$\delta_D([A], 1) = \delta_N(A,1) = [AB]$

$\delta_D([B], 0) = \delta_N(B,0) = [C]$

$\delta_D([B], 1) = \delta_N(B,1) = [C]$

$\delta_D([C], 0) = \delta_N(C,0) = \phi$

$\delta_D([C], 1) = \delta_N(C,1) = \phi$

$\delta_D([AB], 0) = \delta_N(A,0) \cup \delta_N(B,0) = [A] \cup [C] = [AC]$

$\delta_D([AB], 1) = \delta_N(A,1) \cup \delta_N(B,1) = [AB] \cup [C] = [ABC]$

$\delta_D([BC], 0) = \delta_N(B,0) \cup \delta_N(C,0) = [C] \cup \phi = [C]$

$\delta_D([BC], 1) = \delta_N(B,1) \cup \delta_N(C,1) = [C] \cup \phi = [C]$

$\delta_D([AC], 0) = \delta_N(A,0) \cup \delta_N(C,0) = [A] \cup \phi = [A]$

$\delta_D([AC], 1) = \delta_N(A,1) \cup \delta_N(C,1) = [AB] \cup \phi = [AB]$

Scanned with CamScanner

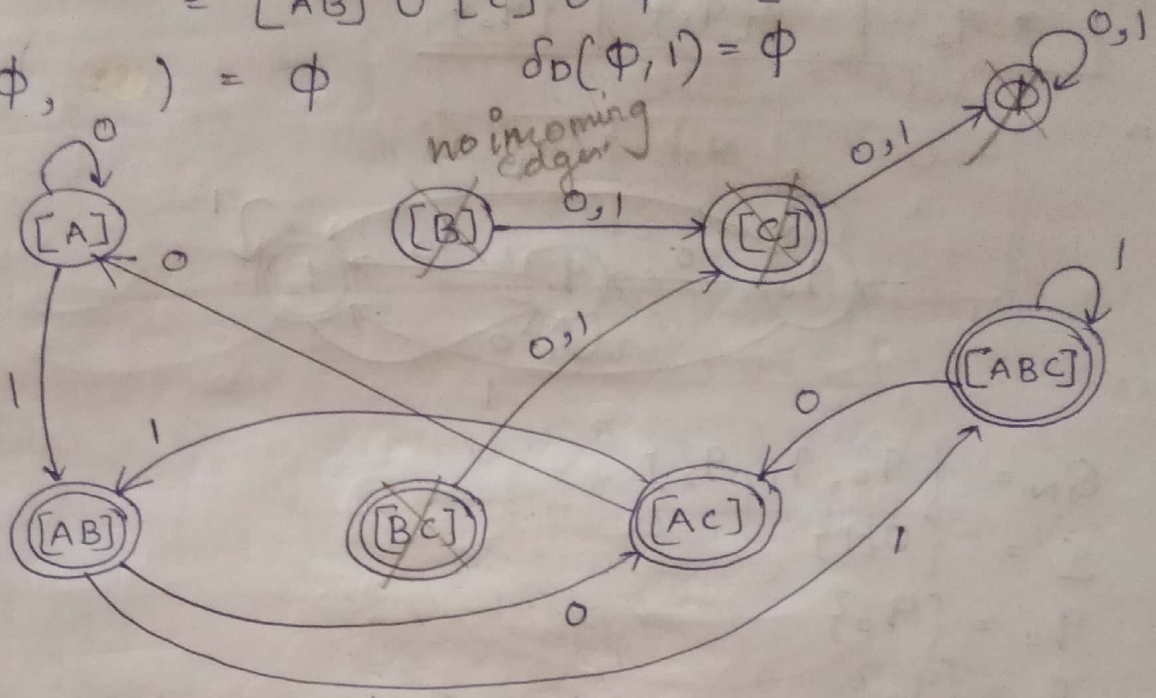
$\delta_D([ABC], 0) = \delta_N(A,0) \cup \delta_N(B,0) \cup \delta_N(C,0)$

$= [A] \cup [C] \cup \phi = [AC]$

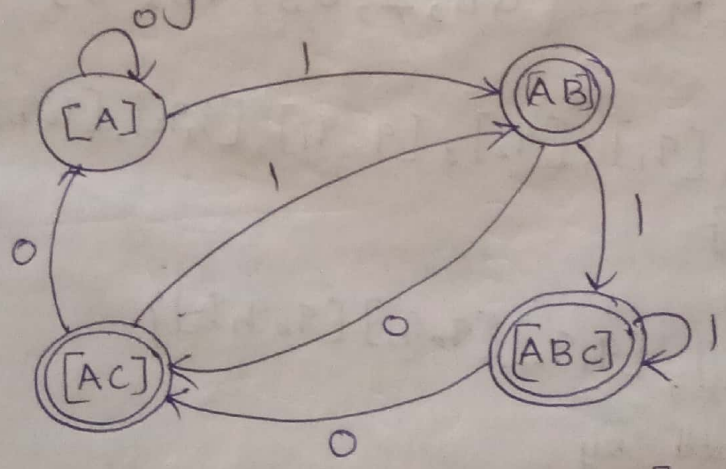
$\delta_D([ABC], 1) = \delta_N(A,1) \cup \delta_N(B,1) \cup \delta_N(C,1)$

$= [AB] \cup [C] \cup \phi = [ABC]$

$\delta_D(\phi, \quad) = \phi$ $\quad \delta_D(\phi, 1) = \phi$

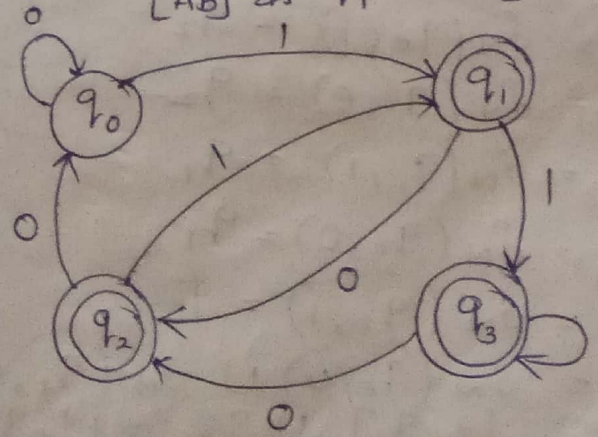


After Eliminating dead states



Renaming state [A] as q_0 [AC] as q_2

[AB] as q_1 [ABC] as q_3

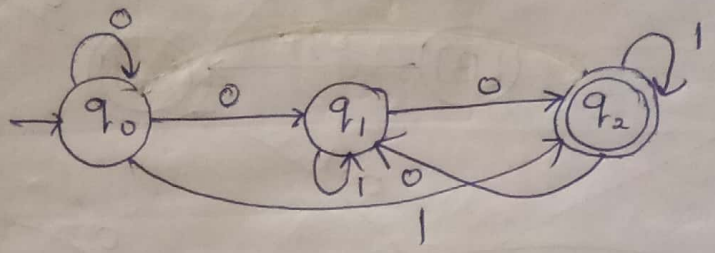


Scanned with CamScanner

2) Construct a DFA from a given NFA.

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

Soln



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

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

$q_0 = \{q_0\}$

$F = \{q_2\}$

The DFA $M = \{Q_D, \Sigma, \delta_D, q_D, F_D\}$ can be defined as

$Q = \{[q_0], [q_1], [q_2], [q_0 q_1], [q_1 q_2], [q_0 q_2], [q_0, q_1, q_2], \phi\}$

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

$q_D = q_0$

$F_D = \{[q_2], [q_1 q_2], [q_0 q_2], [q_0 q_1 q_2]\}$

δ is defined by

$\delta_D(q_0, 0) = \delta_N(q_0, 0) = [q_0 q_1]$

$\delta_D(q_0, 1) = \delta_N(q_0, 1) = q_2$

$\delta_D(q_1, 0) = \delta_N(q_1, 0) = q_2$

$\delta_D(q_1, 1) = \delta_N(q_1, 1) = q_1$

$\delta_D(q_2, 0) = \delta_N(q_2, 0) = q_1$

$\delta_D(q_2, 1) = \delta_N(q_2, 1) = q_2$

$\delta_D(q_0 q_1, 0) = \delta_N(q_0, 0) \cup \delta_N(q_1, 0)$

$= [q_0 q_1] \cup q_2 = [q_0 q_1 q_2]$

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

$= \{q_2\} \cup \{q_1\} = [q_2 q_1]$

Scanned with CamScanner

$\delta_D(q_1q_2,0) = \delta_N(q_1,0) \cup \delta_N(q_2,0)$
$= \{q_2\} \cup \{q_1\} = [q_1q_2]$

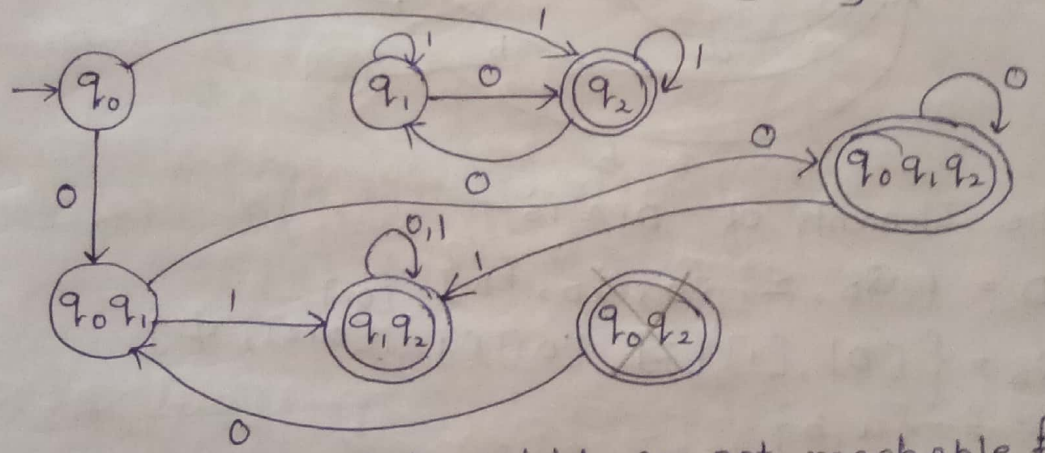
$\delta_D(q_1q_2,1) = \delta_N(q_1,1) \cup \delta_N(q_2,1)$
$= \{q_1\} \cup \{q_2\} = [q_1q_2]$

$\delta_D(q_0q_2,0) = \delta_N(q_0,0) \cup \delta_N(q_2,0)$
$= \{q_0q_1\} \cup \{q_1\} = [q_0q_1]$

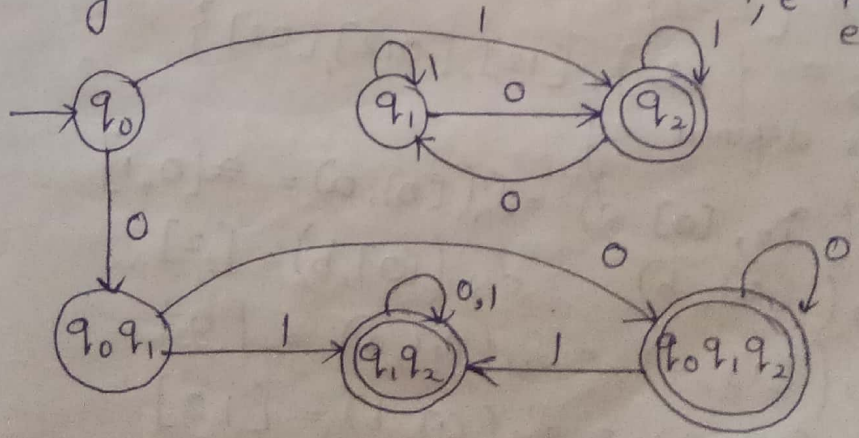
$\delta_D(q_0q_2,1) = \delta_N(q_0,1) \cup \delta_N(q_2,1)$
$= \{q_2\} \cup \{q_2\} = [q_2]$

$\delta_D(q_0,q_1,q_2,0) = \delta_N(q_0,0) \cup \delta_N(q_1,0) \cup \delta(q_2,0)$
$= \{q_0q_1\} \cup \{q_2\} \cup \{q_1\} = [q_0q_1q_2]$

$\delta_D(q_0q_1q_2,1) = \delta_N(q_0,1) \cup \delta_N(q_1,1) \cup \delta_N(q_2,1)$
$= q_2 \cup q_1 \cup q_2 = [q_1q_2]$



After removing the state which are not reachable from q_0, i,e no incoming edges.



Scanned with CamScanner