GATE

Be sure with Vidyalankar Since 1960

Pearl Centre, S.B. Marg, Dadar (W), Mumbai – 400 028. Tel. 4232 4232

CS : Computer Science and Information Technology

Theory of Computation

Index

Topic 1 : Finite State Machines

INTRODUCTION

As the name suggests, the subject "Theory of Computer Science" deals with the theory of automata, Turing Machine, different forms of grammars and much more. You must be thinking, as an engineer why should I learn this subject ? The answer is, for every engineering process there is some mathematical or scientific base, without which no engineer can work. This subject acts as a base for variety of areas of computer engineering such as compiler construction, language processing, operating system design etc.

Basic Concepts

1. **Symbol :**
   Symbol is an abstract or a user defined entity. It cannot be formally defined.

2. **Set :**
   It is a finite set of similar objects.
   A set is determined by its elements. An easy way to describe or specify a finite set is to list all its elements.
   For example

   A = { 1, 2, 3, 4 }

   A precise way of describing a set without listing the elements explicitly is to give a property that characterize the elements.
   For Example

   B = { x | x is an integer greater than 0 and less than 5 }

3. **Alphabets :**
   An alphabet is a finite, nonempty set of symbols.
   Conventionally, we use the symbol Σ for an alphabet.

   Common alphabets include :
   1. $\Sigma = \{0, 1\}$, the binary alphabet
   2. $\Sigma = \{a, b, \ldots z\}$, the set of all lower–case letters

4. **Relation :**
   It is a set of ordered pairs. The first component of the pair is from the set called 'domain' and second component is from the set called 'range'.

   In a relation, if the domain and range are the same set 'S', then we say that the relation is on S.

   If R is a relation and (a, b) is a pair in R, then we write aRb.

Properties of a relation :
If a relation R is on set S, then it is
- i) Reflexive, if aRa exists $\forall\ a \in S$
- ii) Transitive, if aRb and bRc imply aRc
- iii) Symmetric, if aRb implies bRa

5. **Equivalence Relation :**
   If a relation is reflexive, transitive as well as symmetric it is said to be 'Equivalence Relation'.
   For example:
   '=' on set of integers

6. **String :**
   A string over an alphabet Σ is a finite sequence of symbols chosen from alphabet Σ.
   Conventionally strings are denoted by small case letters x, y, z, a, b.
   For Example
   011010 is a string from the binary alphabet $\Sigma = \{0, 1\}$

7. **Empty string :**
   The empty string is the string with zero occurrences of symbols. The string, denoted by $\in$, is a string that may be chosen from any alphabet whatsoever.

8. **Language :**
   A language is defined as a set of valid strings for some alphabet Σ.

Finite State Machine

This machine will consist of an input output relation at every state, and also the changes of the states that will occur on receiving the input at a particular state. Hence we will require the mapping in two forms. i.e. At a particular state, for a given input, what is the output.

$$S \times I \rightarrow O$$

and at a particular state on receiving the input which is the next state we reach

$$S \times I \rightarrow S.$$

These are respectively known as **Machine Function**, MAF, and **State Function**, STF. Both of them are functions of two arguments, current state and current input but their results are different.

The machine can be specified by

1.	A finite set of states	$S = \{q_0, q_1, q_2, \ldots\ldots\ldots\}$
2.	A special element of set S, called as initial state or start state.	
3.	An input alphabet	$I = \{i_0, i_1, \ldots\ldots\ldots\}$
4.	An output alphabet	$O = \{O_0, O_1, \ldots\ldots..\}$
5.	A function	$S \times I \rightarrow S$ (STF)
6.	A function	$S \times I \rightarrow O$ (MAF)

At any state the machine is in any of its states and on arrival of an input symbol it will change the state using the state function and will also give an output using the machine function.

As the number of states is finite, it is known as a finite state machine. This class of machines is also known as Finite Automaton (FA).

At this point we can divide the FSM into two categories, namely,

i. FSM with output and
ii. FSM without output.

The above definition is for the FSM with output. Some times we do not require the output string but the output is just in the form Yes or No or Accept or Reject. We can save some checks during the working of the machine and eliminate the output alphabet as well as output function or machine function. But we will additionally be defining a set of final sates. On receiving the input string, at the end, if we are at any of the final sates, then the string is accepted otherwise it is rejected.

Finite Automaton without output will be defined as a 5 tuple.

FA	=		$(Q, \Sigma, \delta, Q_0, F)$
where	Q	is	Set of states
	Σ	is	Input alphabets
	δ	is	Transitions
			$\delta(Q_k, i) = Q_L$
	Q_0	is	Start state
	F	is	Set of final states

The FSM will be represented in tabular form or by a transition graph.

Design a machine which checks whether a given decimal number is even.

Solution :

Here the number will be accepted digit by digit and every time we assume the number has ended to actually decide whether the number is even or odd. As I give you the number, digit by digit, each time you will be ready with the answer, till the number ends.

There will be 3 states, the number is even, the number is odd, and a start state.

We know that if the current symbol is 0, 2, 4, 6, 8, then we should be in the state indicating that number is even and on any other input we should be in the state indicating that number is odd.

The process continues till the end of the string and if finally we are in the state indicating the number as even, then the given number is even, else odd.

Example: Let us analyse the process in steps, for 10752196

Number up to	Current digit	State
1	1	ODD
10	0	EVEN
107	7	ODD
1075	5	ODD
10752	2	EVEN

and so on

Thus clearly, there will be two states, ODD and EVEN, along with a START state when we do not have any answer i.e. neither EVEN nor ODD.

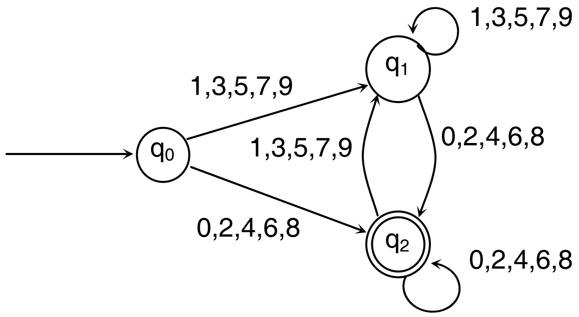
State Table:

State	Input	
	0,2,4,6,8	1,3,5,7,9
q_0	q_2	q_1
odd q_1	q_2	q_1
even $\bigcirc q_2$	q_2	q_1

Therefore

Σ = {0, 1, 2, 3, …}
Q = $\{q_0, q_1, q_2\}$
δ = The transition table shown above
q_0 = Start state
F = $\{q_2\}$

State Diagram:



Suppose the number is 241137836

$q_0 \mid 241137836$	$\delta(q_0, 2) \rightarrow q_2$
$\rightarrow 2q_2 \mid 41137836$	$\delta(q_2, 4) \rightarrow q_2$
$\rightarrow 24q_2 \mid 1137836$	$\delta(q_2, 1) \rightarrow q_1$
$\rightarrow 241q_1 \mid 137836$	$\delta(q_1, 1) \rightarrow q_1$
$\rightarrow 2411q_1 \mid 37836$	$\delta(q_1, 3) \rightarrow q_1$
$\rightarrow 24113q_1 \mid 7836$	$\delta(q_1, 7) \rightarrow q_1$
$\rightarrow 241137q_1 \mid 836$	$\delta(q_1, 8) \rightarrow q_2$
$\rightarrow 2411378q_2 \mid 36$	$\delta(q_2, 3) \rightarrow q_1$
$\rightarrow 24113783q_1 \mid 6$	$\delta(q_1, 6) \rightarrow q_2$
$\rightarrow 241137836q_2$	

As we end up in state q_2, which is a final state, the number is even.

Design a machine to check whether a given decimal number is divisible by 3.

Solution

As we think of divisibility by 3, we may say that if the sum of the digits is divisible by 3 then the number is divisible by 3. This is not the proper way of thinking. We are actually answering in the recursive manner. We are using the problem itself in the solution. The question remains as how will you check whether the sum is divisible by 3. For this purpose we will have to actually see the division technique studied in the school days. We also remember the fact that the last digit or last two digits are not going to tell us whether the number is really divisible by 3 or not.

When we are required to check whether a given number is divisible by 3 we can not decide on the current digit but we are required to check whether the sum of digits is divisible by 3. But it is also not an easy way; we consider the 3 basic states representing the 3 remainders 0, 1, 2. The input will be divided into 3 groups.(0, 3, 6, 9),(1, 4, 7), (2, 5, 8) which will respectively generate remainders 0, 1, 2.

Suppose the number is 2411378362

Remainder	= R
Current Digit	= CD

3	2411378362	
	24 –24	R = 2, CD = 4 Hence R = 0
	01 00	R = 0, CD = 1 Hence R = 1
	11 9	R = 1, CD = 1 Hence R = 2
	023 21	R = 2, CD = 3 Hence R = 2
	27 27	R = 2, CD = 7 Hence R = 0
	08 6	R = 0, CD = 8 Hence R = 2
	23 21	R = 2, CD = 3 Hence R = 2
	26 24	R = 2, CD = 6 Hence R = 2
	22 21	R = 2, CD = 2 Hence R = 2
	1	R = 2, CD = 2 Hence R = 1

Observe that every time the remainder is 0, 1 or 2.

In remainder 1 state when we received the inputs 0, 3, 6, 9, the number generated is 10, 13, 16, 19 which again results in state with remainder 1.

When input is 1, 4, 7 the numbers generated are 11, 14, 17, resulting in remainder 2 and when input is 2, 5, 8, the generated numbers are 12, 15, 18 i.e. state with remainder 0.

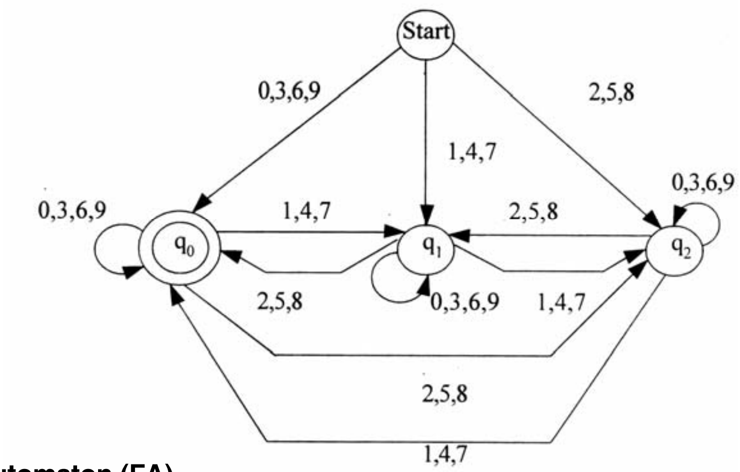
If the same thing is done with state indicating remainder 2, the generated number will give us the next state.

State Table:

	0,3,6,9	1,4,7	2,5,8
Start	q_0	q_1	q_2
Rem 0 q_0	q_0	q_1	q_2
Rem 1 q_1	q_1	q_2	q_0
Rem 2 q_2	q_2	q_0	q_1

q_0 : Yes – Divisible q_1, q_2 : No – Not divisible

Transition Graph :



Finite Automaton (FA)

Finite Automaton (FA) is the mathematical model of a finite–state machine.

Definition :

It is defined by a 5-tuple.

FA = (Q, Σ, δ, q_0, F)

where

Q is a finite set of states

Σ is a finite set of symbols

δ is a transition function that takes as arguments a state and an input symbol and returns a state.

i.e. $\delta : Q \times \Sigma \rightarrow Q$

q_0 is a start state, $q_0 \in Q$

F is a set of final or accepting states, $F \subseteq Q$

Function of FA

A finite Automaton can be constructed for a given language and can be used to accept or reject any string from the given language.

In other words, FA constructed for a language checks the validity of the string for the language.

Finite Control

The FA can be visualized as a finite control with

- i/p string written on tape
- each cell containing only one symbol
- $ denotes end of string
- head or pointer points to current symbol
- it can only be read and that also in one direction i.e. left to right and
- only one cell at a time.

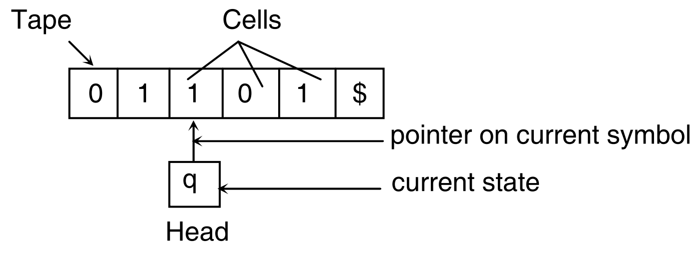


Fig. Finite Control

DETERMINISTIC FINITE AUTOMATON (DFA)

The term 'deterministic" refers to the fact that on each input there is one and only one state to which the automaton can move to from its current state. In other words, given a state q, then no particular input symbol can cause the FA to move into more than one state.

Processing Strings

The first thing we need to know about a DFA is how the DFA decides whether or not to "accept" a sequence of input symbols. The "language" of the DFA is the set of all strings that the DFA accepts. Suppose $a_1a_2....a_n$ is a sequence of input symbols. We start out with the DFA in its start state, q_0. We consult the transition function δ, say $\delta(q_0, a_1) = q_1$ to find the state that the DFA enters after processing the first input symbol a_1. We process the next input symbol, a_2 by evaluating $\delta(q_1, a_2)$; let us suppose this state is q_2. We continue in this manner, finding states $q_3, q_4,.....,q_n$ such that $\delta(q_{i-1}, a_i) = q_i$ for each i. If q_n is a member of F, then the input $a_1a_2......a_n$ is accepted, and if not then it is rejected.

Notations for DFA

Specifying a DFA as a five-tuple with detailed description of the δ-transition function is both tedious and hard to read. There are two preferred notations for describing automata:
1. A transition diagram, which is a graph.
2. A transition table, which is a tabular listing of the δ function.

Transition Diagram

A transition diagram for a DFA = $(Q, \Sigma, \delta, q_0, F)$ is a graph defined as follows :

a) For each state q in Q there is a node.
b) For each state q in Q and each input symbol a in Σ, let $\delta(q, a) = p$. Then the transition diagram has an arc from node q to node p, labeled a. If there are several input symbols that cause transitions from q to p, then the transition diagram can have one arc, labeled by the list of these symbols.
c) There is an arrow into the start state q_0, labeled start.
   This arrow does not originate at any node.
d) Nodes corresponding to accepting states (those in F) are marked by a double circle. States not in F have a single circle.

Transition Tables

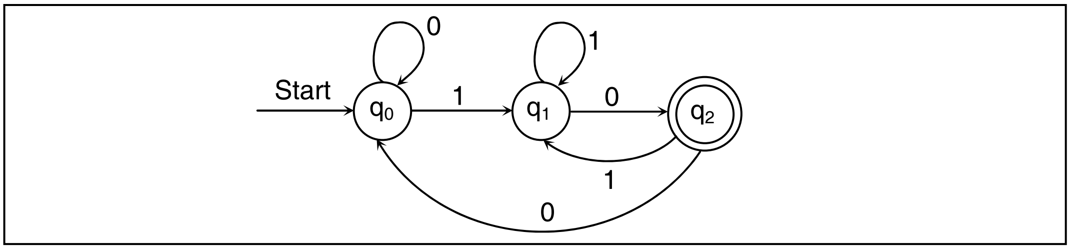
A transition table is a conventional, tabular representation of a function like δ that takes two arguments and returns a value. The rows of the table correspond to the states, and the columns correspond to the inputs. The entry for the row corresponding to state q and the column corresponding to input 'a' is the state $\delta(q, a)$.

The final states in a transition table are represented by marking them with a circle.

Consider a FA that recognizes all binary strings ending in 10.
Here, $\Sigma = \{0, 1\}$
We will construct the FA that will accept the given language i.e. binary strings ending in 10.

If we get a zero while in state q_0, we remain in the same state and on getting a one, we move to state q_1. While in q_1, we stay in the same state till we get a zero, which leads us to state q_2.

Transition Diagram



Transition Table

Inputs / States	0	1
start $\rightarrow$ q_0	**q_0**	q_1
q_1	**q_2**	q_1
$\textcircled{q_2}$	q_0	q_1

Extended Transition Function

We have explained informally that the DFA defines a language : the set of all strings that result in a sequence of state transitions from the start state to an accepting state. In term of the transition diagram, the language of a DFA is the set of labels along all paths that lead from the start state to any accepting state.

Now, we will make the notation of the language of a DFA precise. To do so, we define an 'Extended Transition Function' that describes what happens when we start in any state and follow any sequence of inputs. If δ is our transition function, then the extended transition function constructed from δ will be called $\hat{\delta}$. The extended transition function is function that takes a state q and a string ω and returns a state p – the state that the automaton reaches when starting in state q and processing the sequence of inputs w.

Definition :
Let A = (Q, Σ, δ, q_0, F) be an FA. We define the function
$\hat{\delta} : Q \times \Sigma^* \to Q$
as follows :
1. For any q ∈ Q, $\hat{\delta}(q, \varepsilon) = q$
2. For any q ∈ Q, y ∈ Σ^*, and a ∈ Σ $\hat{\delta}(q, ya) = \delta(\hat{\delta}(q, y), a)$

The Language of a DFA

Now, we can define the language of a DFA A = (Q, Σ, δ, q_0, F). This language is denoted L(A), and is defined by $L(A) = \{w \mid \hat{\delta}(q_0, w) \text{ is in } F\}$
That is, the language of A is the set of strings w that take the start state q_0 to one of the accepting states. If L is L(A) for some DFA, then we say L is 'regular language'.

NON-DETERMINISTIC FINITE AUTOMATON

For some cases, finding a finite automaton (FA) corresponding to a given regular expression can be tedious and unintuitive if we rely only on the techniques we have developed so far, which involve deciding at each step how much information about the input string it is necessary to remember. An alternative approach is to consider a formal device called a Non–deterministic Finite Automata (NFA), similar to an FA but with the rules relaxed somewhat. Constructing NFA to correspond to a given regular expression is often much simpler. Furthermore, it will turn out that NFAs accept exactly the same languages as FAs, and there are straightforward procedures for converting an NFA to an equivalent FA.
A "non-deterministic" finite automaton (NFA) has the power to be in several states at once. That is, an NFA has more than one transition on the same i/p symbol from some state.

Definition :
A non-deterministic finite automaton, abbreviated NFA, is a 5-tuple A = {Q, Σ, δ, q_0, F} where Q and Σ are non empty finite sets, q_0 ∈ Q, A ⊆ Q, and
$\delta : Q \times \Sigma \to 2^Q$
Q is the set of states, Σ is the alphabet, q_0 is the initial state, and A is the set of accepting states.

Note :
δ is the transition function that takes a state in Q and an input symbol in Σ as argument and returns a subset of Q. Notice that the only difference between an NFA and DFA is in the type of value the δ returns : a set of states in the case of an NFA and a single state in the case of a DFA.

Illustration :
The figure below shows a non-deterministic finite automaton, whose job is to accept all and only the strings of 0's and 1's that end in 01.

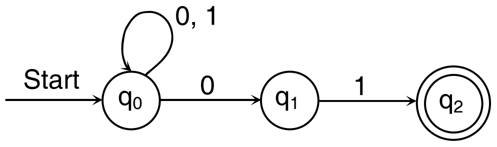


Fig. An NFA accepting all strings that end in 01

State q_0 is the start state, and we can think of the automaton as being in state q_0 whenever it has not yet 'guessed' that the final 01 has begun. It is always possible that the next symbol does not begin the final 01, even though that symbol is 0. Thus, state q_0 may transition to itself on both 0 and 1.

However, if the next symbol is 0, this NFA also guesses that the final 01 has begun. An arc labeled 0 thus leads from q_0 to state q_1.

Let us now consider the ways in which the transition diagram of an NFA differs from that of an FA. There are apparently two major differences. From some states there are no transitions for both input symbols (from state q_2, there are no transitions at all); and from some states there is more than one transition for the same input symbol (from state q_0, three are two arrows corresponding to 0).

The way to interpret the first of these features in easy. The absence of an a-transition from state q means that from q there is no input string beginning with 'a' that can result in the device's being in an accepting state. We could create a transition by creating "dead" state to which the device could go from q on input a, having the property that once the device gets to that state it can never leave it. However, leaving this transition out makes the picture simpler, and because it would never be executed during any sequence of moves leading to an accepting state, leaving it out does not hurt anything except that it violates the rules.

The second violation of the rules for FAs seems to be more serious. The above figure shows two transitions from q_0 on input 0. It does not specify an unambiguous action for that state-input combination, and therefore apparently no longer represents a recognition algorithm or a language-recognizing machine. But we will see shortly that for every NFA there exists an FA that operates in such a way that it simulates the NFA correctly and accepts the right language.

The NFA of the above figure can be specified formally as

$$(\{q_0, q_1, q_2\},\ \{0, 1\},\ \delta, q_0, \{q_2\})$$

States \ Inputs	0	1
start $\rightarrow$ q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	ϕ	$\{q_2\}$
$\boxed{q_2}$	ϕ	ϕ

Fig. Transition table for an NFA that accepts all strings ending in 01

Notice that transition tables can be used to specify the transition function for an NFA as well as for a DFA. The only difference is that each entry in the table for the NFA is a set, even if the set is a singleton (has one member). Also notice that when there is no transition at all from a given state on a given input symbol, the proper entry is ϕ, the empty set.

Extended Transition Function

As for DFA's, we need to extend the transition function δ of an NFA to a function $\hat{\delta}$ that takes a state q and string of input symbols w, and returns the set of states that the NFA is in if it starts in state q and processes the string w.

Definition :

Let A = {Q, Σ, δ, q_0, F) be an NFA. The function $\hat{\delta} : Q \times \Sigma^* \rightarrow 2^Q$ is defined as follows :

1. For any $q \in Q$, $\hat{\delta}(q, \varepsilon) = \{q\}$
2. For any $q \in Q$, $y \in \Sigma^*$ and $a \in \Sigma$

$$\hat{\delta}(q, ya) = \bigcup_{r \in \hat{\delta}(q, y)} \delta(r, a)$$

The Language of an NFA :

A string w is accepted by an NFA means that there is a sequence of moves it can make, starting in its initial state and processing the symbols of w, that will lead to an accepting state. In other words, NFA accepts w if the set of states in which it can end up as a result of processing w contains at least one accepting state. The fact that other choices using the input symbols of w lead to an non-accepting state, or do not lead to any state at all (i.e. sequence of states "dies"), does not prevent w from being accepted by the NFA as a whole.

Formally, if A = (Q, Σ, δ, q_0, F) is an NFA, then

$$L(A) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \phi\}$$

That is, L(A) is the set of strings w in Σ^* such that $\hat{\delta}(q_0, w)$ contains at least one accepting state.

Equivalence of Deterministic and Non-deterministic Finite Automata

NFA and DFA are equivalent to each other. In other words, for every NFA there exists an equivalent DFA accepting the same set of words. Therefore, the capabilities of NFA and its DFA are same. But there are many languages for which an NFA is easier to construct than a DFA. Moreover, the DFA in practice has about as many states as the NFA, although it often has more transitions. In the worst case, however, the smallest DFA can have 2^n states while the smallest NFA for the same language has only n states.

NFA and DFA Conversion

The NFA to DFA conversion involves an important "construction" called the subset construction because it involves constructing all subsets of the set of states of the NFA.

The subset constructions start from an NFA N = (Q, Σ, δ, q_0, F). Its goal is the description of DFA D = $(Q', \Sigma, \delta', \{q_0\}, F')$ such that L(D) = L(N). Note that the input alphabets of the two automata are the same, and the start state of D is the set containing only the start state of N. The other components of D are constructed as follows :

1. The major difference between NFA and DFA is because of the transition function δ of the NFA. Because of this the entries in the state table are sets instead of singular entries like in DFA.
   Q' is the set of subsets of Q; i.e. Q' is the power set of Q. While converting an NFA into an equivalent DFA we consider Q' having 2^n states if Q has n states.
   ∴ The state function will become,
   $$\delta : Q' \times \Sigma \to Q'$$

2. F' is the set of subsets S of Q such that $S \cap F \neq \phi$. That is, F' consists of all sets of NFA's states that include atleast one accepting state of NFA.

3. For each set $S \subseteq Q$ and for each input symbol a in Σ,
   $$\delta'(S, a) = \bigcup_{p \text{ in } S} \delta(p, a)$$
   That is, to compute $\delta'(S, a)$ we look at all states p in S, see to what states the NFA goes from p on input a, and take the union of all those states.
   Now, we will convert the NFA given below into a DFA.
   $N = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$
   where, δ is

δ	0	1
q_0	$\{q_0, q_1\}$	$\{q_1\}$
$\boxed{q_1}$	ϕ	$\{q_0, q_1\}$

Now, for given NFA N,
$Q = \{q_0, q_1\}$, $\Sigma = \{0, 1\}$, $F = q_1$ and
initial state is q_0

We can denote the resultant DFA as,
$D = \{Q', \Sigma, \delta', \{q_0\}, F'\}$

where Σ = {0, 1} and initial state is q_0 itself but it is represented by a singleton $\{q_0\}$.

Now power set Q = 2^Q = $(\phi, \{q_0\}, \{q_1\}, \{q_0, q_1\})$. But we will exclude φ as this state does appear on L.H.S. of δ i.e. it will be inaccessible even if it is included.

∴ $Q' = (\{q_0\}, \{q_1\}, \{q_0, q_1\})$

Now, we will find the state function of equivalent DFA D.

i.e. $\delta' : Q' \times \Sigma \to Q'$

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

$\delta' = (\{q_0\}, 1) \quad = \delta(q_0, 1) = \{q_1\}$

$\delta' = (\{q_1\}, 0) \quad = \delta(q_1, 0) = \phi$

$$
\begin{aligned}
\delta' &= (\{q_1\}, 1) &&= \delta(q_1, 1) = \{q_0, q_1\} \\
\delta' &= (\{q_0, q_1\}, 0) &&= \delta(q_0, 0) \cup \delta\{q_1, 0\} \\
& &&= \{q_0, q_1\} \cup \phi \\
& &&= \{q_0, q_1\} \\
\delta'&(\{q_0, q_1\}, 1) &&= \delta(q_0, 1) \cup \delta(q_1, 1) \\
& &&= \{q_1\} \cup \{q_0, q_1\} \\
& &&= \{q_0, q_1\}
\end{aligned}
$$

$\therefore$ From above δ', transition table can be constructed as

δ'	0	1
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_1\}$
$\{q_1\}$	–	$\{q_0, q_1\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_1\}$

$\{q_1\}$ and $\{q_0, q_1\}$ are final states as they include the final state q_1 of the NFA

Transition Diagram

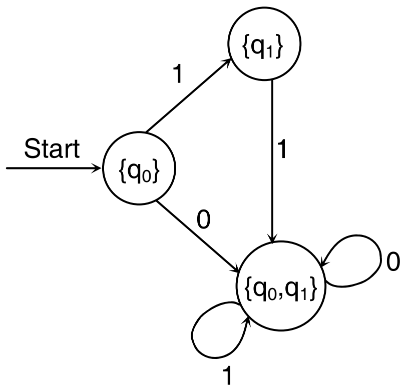


Fig. Equivalent DFA

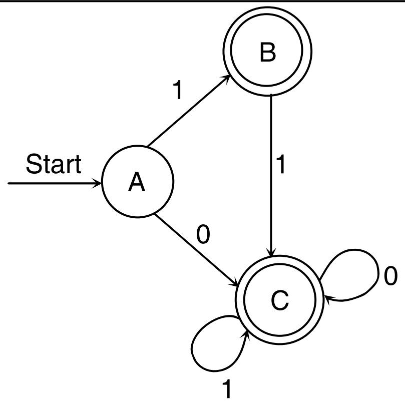


Fig. After Relabeling

Final Transition Table

δ'	0	1
A	C	B
B	–	C
C	C	C

$\therefore$ $Q' = \{A, B, C\}$, $\Sigma = \{0, 1\}$

$q_0 = A$ and $F' = \{B, C\}$

Direct method

I) From transition diagram to transition diagram.
Consider the NFA whose transition diagram is given below

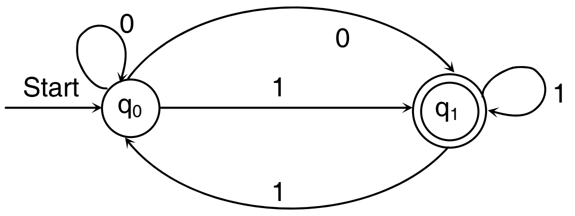


Fig. NFA

Procedure : First process initial state q_0

Processing q_0 :

We can see that 'q_0' on '0' goes to 'q_0' and 'q_1' both. Therefore, create new state labeled 'q_0q_1'. As this new state contains the label 'q_1' which is the final state of given NFA, mark this state as final state. See figure (a) given below.

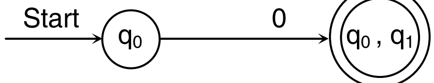


Fig. (a)

'q_0' on '1' goes to only 'q_1'. Therefore, add one more state 'q_1' as shown in figure (b).

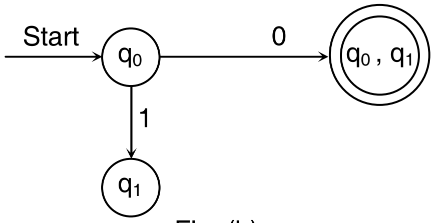


Fig. (b)

Now state q_0 is processed for all possible inputs. Now similarly process other states.

Processing q_1 :

q_1 on 0 goes nowhere. Therefore, no transition from q_1 is labeled by '0'.

q_1 on 1 goes to q_1 and q_0. Therefore, show transition from 'q_1' to 'q_0q_1' which is already existing (If it is not existing then create a new state.)

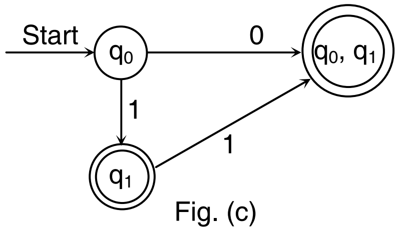


Fig. (c)

Processing q_0q_1 :

$$\begin{aligned}\delta'(q_0q_1, 0) &= \delta(q_0, 0) \cup \delta(q_1, 0)\\ &= \{q_0, q_1\} \cup \phi\\ &= \{q_0, q_1\}\end{aligned}$$

and

$$\begin{aligned}\delta'(q_0q_1, 1) &= \delta(q_0, 1) \cup \delta(q_1, 1)\\ &= \{q_1\} \cup \{q_0, q_1\}\\ &= \{q_0, q_1\}\end{aligned}$$

$\therefore$ $\{q_0, q_1\}$ on 0 and 1 goes to itself.

Now, all the states are processed.

Therefore, the final transition diagram is,

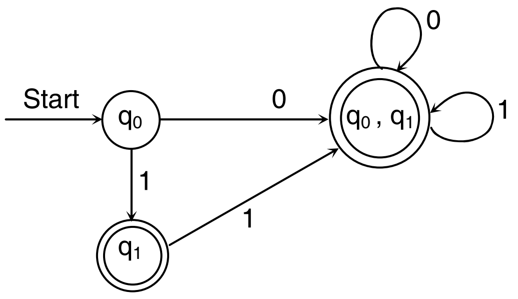


Fig. Transition Diagram

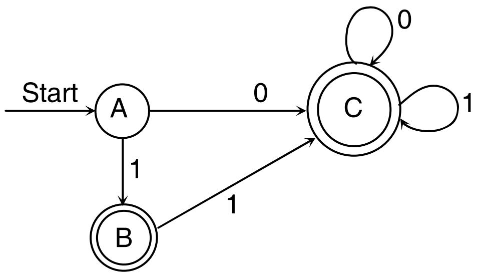


Fig. Transition Diagram after Relabeling

Transition Table

δ	0	1
A	C	B
$\textcircled{B}$	–	C
$\textcircled{C}$	C	C

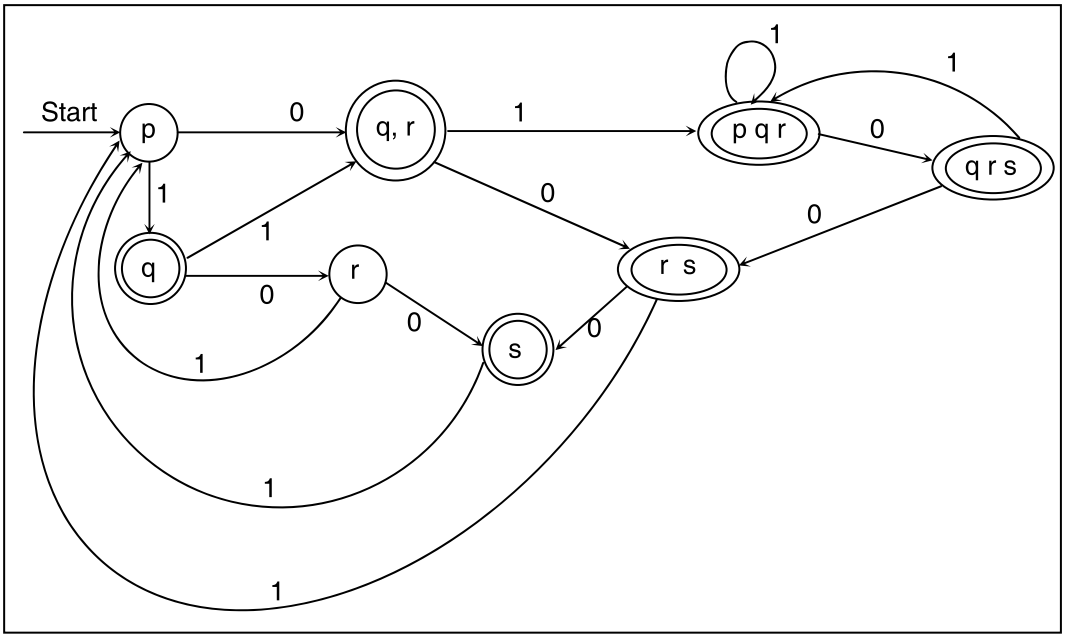
II. From Transition Table to Transition Diagram
Consider the NFA given below

({p, q, r, s}, {0, 1}, δ, p, {q, s})

and δ is

δ	0	1
p	q, r	q
$\textcircled{q}$	r	q, r
r	s	p
$\textcircled{s}$	–	p

Transition diagram of the equivalent DFA



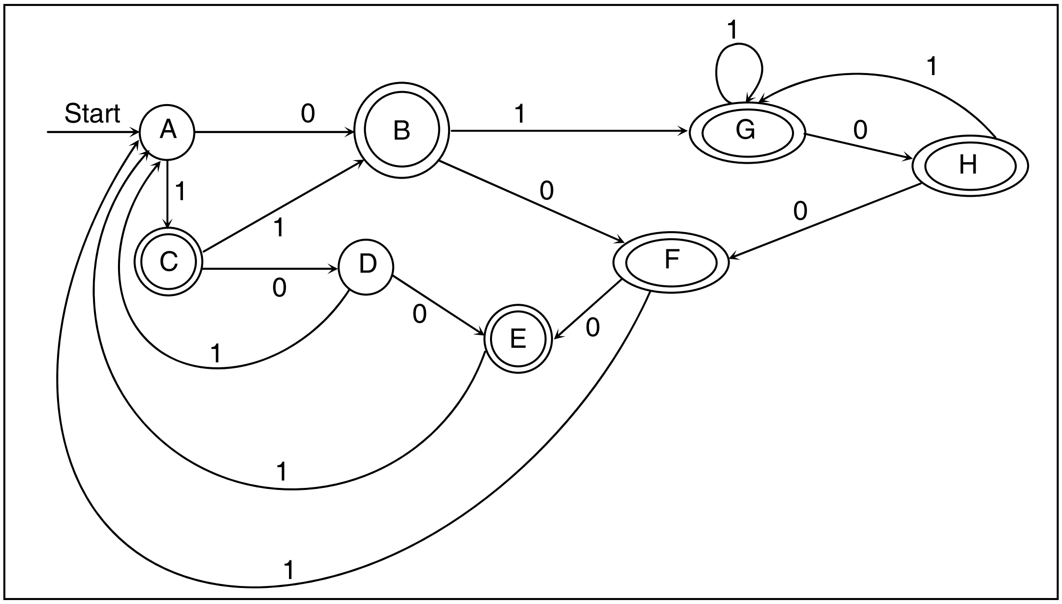
Example

$$\delta'(pqr, 0) = \delta(p, 0) \cup \delta(q, 0) \cup \delta(r, 0)$$
$$= \{q, r\} \cup \{r\} \cup \{s\}$$
$$= \{q, r, s\}$$

$$\delta'(pqr, 1) = \delta(p, 1) \cup \delta(q, 1) \cup \delta(r, 1)$$
$$= \{q\} \cup \{q, r\} \cup \{p\}$$
$$= \{p, q, r\}$$

$$\delta'(pqr, 1) = \delta(p, 1) \cup \delta(q,r,\ 1)$$
$$= \{q\} \cup \{p, q, r\}$$
$$= \{p, q, r\}$$

Relabeling :



Transition Table :

δ′	0	1
A	B	C
Ⓑ	F	G
Ⓒ	D	B
D	E	A
Ⓔ	–	A
Ⓕ	E	A
Ⓖ	H	G
Ⓗ	F	G

Minimization

State equivalence :

When two states S_i and S_j excited by the same input sequences yield identical o/p sequences ; then these two states are said to be equivalent, otherwise they are distinguishable.

Rules for minimization

1. We can replace one final state by its equivalent final state only.
2. We can replace one non-final state by its equivalent non-final state only.
3. We cannot replace initial state by any other states.
4. We cannot replace one final state by a non-final state or vice-versa.
5. Replacing state ‘A’ by state ‘B’ means
   - deleting entries related to ‘A’ i.e. transitions from A
   - and from the table, wherever we find symbol ‘A’ replace it by symbol ‘B’.

Even similar columns can be merged in cases where we have similar transitions for all the states or different inputs.

B and H can be merged as both are final states.
$\therefore$ Remove H and replace H by B in table

δ'	0	1
A	B	C
$\textcircled{B}$	F	G
$\textcircled{C}$	D	B
D	E	A
$\textcircled{E}$	–	A
$\textcircled{F}$	E	A
$\textcircled{G}$	$B^{\bullet}$	G

$\bullet \Rightarrow$ changed state

III From Transition Table to Transition Table
Convert the given NFA to DFA

δ	a	b
Q_0	q_1	q_2
Q_1	q_0, q_3	q_1
Q_2	q_2	q_3
$\textcircled{q_3}$	q_1	q_1, q_2

DFA :

	δ'	a	b
	q_0	q_1	q_2
	q_1	q_4	q_1
	q_2	q_2	q_3
	$\textcircled{q_3}$	q_1	q_5
q_0, q_3	$\textcircled{q_4}$	q_1	q_5
q_1, q_2	q_5	q_6	q_7
q_0, q_2, q_3	$\textcircled{q_6}$	q_5	q_8
q_1, q_3	$\textcircled{q_7}$	q_9	q_5
q_1, q_2, q_3	$\textcircled{q_8}$	q_{10}	q_8
q_0, q_1, q_3	$\textcircled{q_9}$	q_9	q_5
q_0, q_1, q_2, q_3	$\textcircled{q_{10}}$	q_{10}	q_8

* All states containing q_3 are final states.
*
- q_7 and q_9 can be merged
- q_8 and q_{10} can be merged

δ'	a	b
q_0	q_1	q_2
q_1	q_4	q_1
q_2	q_2	q_3
q_3	q_1	q_5
q_4	q_1	q_5
q_5	q_6	q_7
q_6	q_5	q_8
q_7	$q_{\dot{7}}$	q_5
q_8	$q_{\dot{8}}$	q_8

Partition method

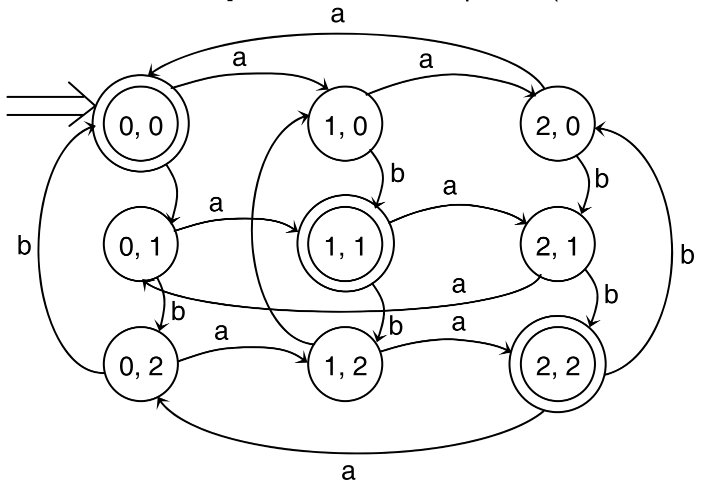
Let N = (Q_N, $\in$, δ_N, q_0, f_N) be DFA. We will consider only set of states of DFA(Q_N) and apply a partition on it as follows.

1. Let π = Q_N, firstly divide Q_N into partitions π_0 with two block one contains all final states and another contains all non-final states.
2. Now, for every pair of states in one block in π_0, for every input symbol, check the transitions. If those two states give the transition in same block then keep them in same group in which they are otherwise group them in two separate groups.
3. Repeat this procedure till you are not able to group the states into further groups.
4. Let π has been divided into separate n groups. These all will act an individual state in the resulting automata.
5. Transitions can be specified by using the transitions of specific state in that particular group to the state present in same or another group.

Let us go through example.

a/mod3 − b/mod3 = 0

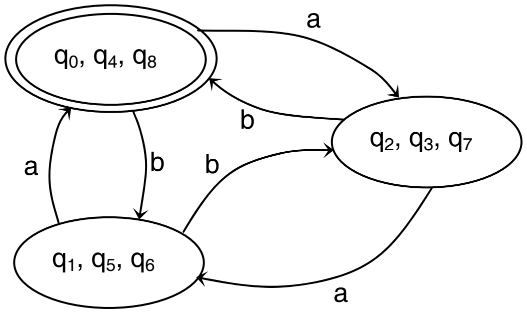
The automata that we can design will have 9 states. Because mod 3 suggest the reminders could be 0, 1, 2. So for a and b we can have 3 possibilities each. So all in all we have 3 × 3 i.e. 9 states & automata as [Let us take ordered pair as (a/mod3, b/mod3)].



Transition table:

(a, b)	a	b
$*q_0(0, 0)$	q_3	q_1
$q_1(0, 1)$	q_4	q_2
$q_2(0, 2)$	q_5	q_0
$q_3(1, 0)$	q_6	q_4
$*q_4(1,1)$	q_7	q_5
$q_5(1, 2)$	q_8	q_3
$q_6(2, 0)$	q_0	q_7
$q_7(2, 1)$	q_1	q_8
$*q_8(2, 2)$	q_2	q_6

Applying algorithm,
q states DFA, can be partitioned as
$\pi_0 = \{(q_0, q_4, q_8), (q_1, q_2, q_3, q_5, q_6, q_7)\}$
Now, for first block (q_0, q_4, q_8) on input a & b the transitions goes to second block. So this block will not be divided.
Now, for block $(q_9, q_2, q_3, q_5, q_6, q_7)$, q_1 on input a goes to q_4 (block 1) and q_2 on input a goes to q_5 (block 2). So clearly these two states will be in two different block. Applying the same method, we will get
$\pi_1 = \{(q_0, q_4, q_8), (q_1, q_5, q_6), (q_2, q_3, q_7)\}$
Now, we will get
$\pi_2 = \pi_1$
So we will stop here and automata will be as



EQUIVALENCE AND MINIMIZATION OF AUTOMATA

The question of whether two descriptions of two regular languages actually define the same language involves considerable intellectual mechanics. In this section we discuss how to test whether two descriptors for regular languages are equivalent, in the sense that they define the same language. An important consequence of this test is that there is a way to minimize a DFA. That is, we can take any DFA and find an equivalent DFA that has minimum number of states. In fact, this DFA is essentially unique: given any two minimum – state DFA's that are equivalent, we can always find a way to rename the states so that the two DFA's become the same.

Testing Equivalence of States

We shall begin by asking a question about the states of a single DFA. Our goal is to understand when two distinct states p and q can be replaced by a single state that behaves like both p and q. We say that states p and q are equivalent if :

- For all input strings w, $\hat{\delta}(p,w)$ is an accepting state if and only if $\hat{\delta}(q,w)$ is an accepting state.

Less formally, it is impossible to tell the difference between equivalent states p and q merely by starting in one of the states and asking whether or not a given input string leads to acceptance when the automaton is started in this (unknown) state. Note we do not require that $\hat{\delta}(p,w)$ and $\hat{\delta}(q,w)$ are the same state, only that either both are accepting or both are nonaccepting.

If two states are not equivalent, then we say they are distinguishable. That is, state p is distinguishable from state q if there is at least one string w such that one of $\hat{\delta}(p,w)$ and $\hat{\delta}(q,w)$ is accepting and the other is not accepting.

Example:

Consider the DFA of figure below, whose transition function we shall refer to as δ in this example. Certain pairs of states are obviously not equivalent. For example, C and G are not equivalent because one is accepting and the other is not. That is, the empty string distinguishes these two states, because $\hat{\delta}(C,\varepsilon)$ is accepting and $\hat{\delta}(G,\varepsilon)$ is not.

Consider states A and G. String ε doesn't distinguish them, because they are both nonaccepting states. String 0 doesn't distinguish them because they go to states B and G, respectively on input 0 and both these states are nonaccepting. Likewise, string 1 doesn't distinguish A from G, because they go to F and E, respectively and both are nonaccepting. However, 01 distinguishes A from G, because $\hat{\delta}(A,01)=C, \hat{\delta}(G,01)=E, C$ is accepting and E is not. Any input string that takes A and G to states only one of which is accepting is sufficient to prove that A and G are not equivalent.

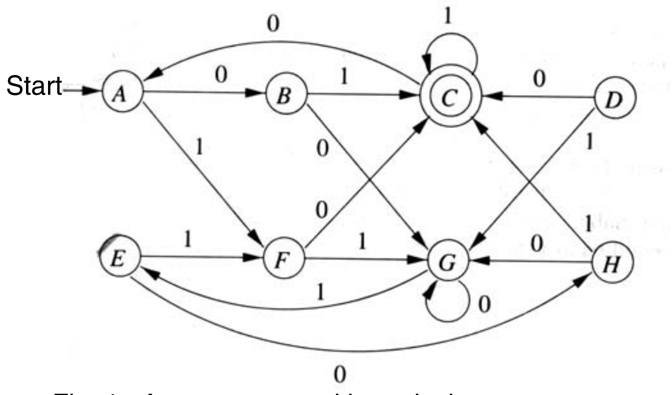


Fig. 1 : An automaton with equivalent states

In contrast, consider states A and E. Neither is accepting, so ε does not distinguish them. On input 1, they both go to state F. Thus, no input string that begins with 1 can distinguish A from E, since for any string x, $\hat{\delta}(A, 1x) = \hat{\delta}(E, 1x)$.

Now consider the behavior of states A and E on inputs that begin with 0. They go to states B and H, respectively. Since neither is accepting, string 0 by itself does not distinguish A from E. However, B and H are no help. On input 1 they both go to C and on input 0 they both go to G. Thus, all inputs that begin with 0 will fail to distinguish A from E. We conclude that no input string whatsoever will distinguish A from E; i.e., they are equivalent states.

To find states that are equivalent, we make our best efforts to find pairs of states that are distinguishable. It is perhaps surprising, but true, that if we try our best, according to the algorithm to be described below, then any pair of states that we do not find distinguishable are equivalent. The algorithm, which we refer to as the table-filling algorithm, is a recursive discovery of distinguishable pairs in a DFA A = $(Q, \Sigma, \delta, q_0, F)$.

BASIS: If p is an accepting state and q is nonaccepting, then the pair {p, q} is distinguishable.

INDUCTION: Let p and q be states such that for some input symbol a, r = δ (p, a) and s = δ (q, a) are a pair of states known to be distinguishable. Then {p, q} is a pair of distinguishable states. The reason this rule makes sense is that there must be some string w that distinguishes r from s; that is, exactly one of $\hat{\delta}(r, w)$ and $\hat{\delta}(s, w)$ is accepting. Then string aw must distinguish p from q, since $\hat{\delta}(p, aw)$ and $\hat{\delta}(q, aw)$ is the same pair of states as $\hat{\delta}(r, w)$ and $\hat{\delta}(s, w)$.

Example :

Let us execute the table-filling algorithm on the DFA of figure 1. The final table is shown in figure below, where an x indicates pairs of distinguishable states and the blank squares indicate those pairs that have been found equivalent. Initially, there are no x's in the table.

B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x
	A	B	C	D	E	F	G

Fig.2 : Table of state inequivalences

For the basis, since C is the only accepting state, we put x's in each pair that involves C. Now that we know some distinguishable pairs, we can discover others. For instance, since {C, H} is distinguishable and states E and F goto H and C, respectively, on input 0, we know that {E, F} is also a distinguishable pair. In fact, all the x's in above figure with the exception of the pair {A, G} can be discovered simply by looking at the transitions from the pair of states on either 0 or on 1 and observing that (for one of those inputs) one state goes to C and the other does not. We can show {A,G} is distinguishable on the next round, since on input 1 they go to F and E, respectively and we already established that the pair {E, F} is distinguishable.

However, then we can discover no more distinguishable pairs. The three remaining pairs, which are therefore equivalent pairs, are {A, E}, {B,H}, and {D,F}. For example, consider why we can not infer that {A,E} is a distinguishable pair. On input 0, A and E go to B and H, respectively, and {B,H} has not yet been shown distinguishable. On input 1, A and E both go to F, so there is no hope of distinguishing them that way. The other two pairs, {B,H} and {D,F} will never be distinguished because they each have identical transitions on 0 and identical transitions on 1. Thus, the table-filling algorithm stops with the table as shown in the figure above, which is the correct determination of equivalent and distinguishable states.

Theorem :
If two states are not distinguished by the table-filling algorithm, then the states are equivalent.

Proof :

Let us again assume we are talking of the DFA A = $(Q, \Sigma, \delta, q_0, F)$. Suppose the theorem is false; that is, there is at least one pair of states {p, q} such that

1. States p and q are distinguishable, in the sense that there is some string w such that exactly one of $\hat{\delta}(p, w)$ and $\hat{\delta}(q, w)$ is accepting and yet
2. The table-filling algorithm does not find p and q to be distinguished.

Call such a pair of states a bad pair.
If there are bad pairs, then there must be some that are distinguished by the shortest strings among all those strings that distinguish bad pairs. Let {p, q} be one such bad pair and let $w = a_1\ a_2\ \ldots.\ a_n$ be a string as short as any that distinguishes p from q. Then exactly one of $\hat{\delta}$ (p, w) and $\hat{\delta}$ (q, w) is accepting.

Observe first that w cannot be ε, since if ε distinguishes a pair of states, then that pair is marked by the basis part of the table-filling algorithm. Thus, $n \geq 1$.

Consider the states $r = \delta\ (p, a_1)$ and $s = \delta\ (q, a_1)$. States r and s are distinguished by the string $a_2a_3\ \ldots.a_n$, since this string takes r and s to the states $\hat{\delta}$ (p, w) and $\hat{\delta}$ (q, w). However, the string distinguishing r from s is shorter than any string that distinguishes a bad pair. Thus, {r, s} cannot be a bad pair. Rather, the table-filling algorithm must have discovered that they are distinguishable.

But the inductive part of the table-filling algorithm will not stop until it has also inferred that p and q are distinguishable, since it finds that $\delta\ (p, a_1) = r$ is distinguishable from $\delta\ (q, a_1) = s$. We have contradicted our assumption that bad pairs exist. If there are no bad pairs, then every pair of distinguishable states is distinguished by the table-filling algorithm and the theorem is true.

Testing Equivalence of Regular Languages

The table-filling algorithm gives us an easy way to test if two regular languages are the same. Suppose languages L and M are each represented in some way, e.g., one by a regular expression and one by an NFA. Convert each representation to a DFA. Now, imagine one DFA whose states are the union of the states of the DFA's for L and M. Technically, this DFA has two start states, but actually the start state is irrelevant as far as testing state equivalence is concerned, so make any state the lone start state.

Now, test if the start states of the two original DFA's are equivalent, using the table-filling algorithm. If they are equivalent, then L = M and if not, then L $\neq$ M.

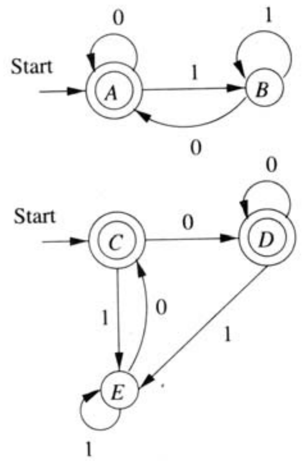


Fig. 3 : Two equivalent DFA's

Example :

Consider the two DFA's in above figure. Each DFA accepts the empty string and all strings that end in 0; that is the language of regular expression $\varepsilon + (0 + 1)^* 0$. We can imagine that above figure represents a single DFA, with five states A through E. If we apply the table – filling algorithm to that automaton, the result is as shown in following figure.

B	x			
C		x		
D		x		
E	x		x	x
	A	B	C	D

Figure 4: The table of distinguishabilities for figure 3.

To see how the table is filled out, we start by placing x's in all pairs of states where exactly one of the states is accepting. It turns out that there is no more to do. The four remaining pairs, {A, C}, {A, D}, {C, D}, and {B, E} are all equivalent pairs. You should check that no more distinguishable pairs are discovered in the inductive part of the table-filling algorithm. For instance, with the table as in above figure, we cannot distinguish the pair {A, D} because on 0 they go to themselves and on 1 they go to the pair {B,E}, which has not yet been distinguished. Since A and C are found equivalent by this test and those states were the start states of the two original automata, we conclude that these DFA's do accept the same language.

The time to fill out the table and thus to decide whether two states are equivalent is polynomial in the number of states. If there are n states, then there are $\binom{n}{2}$ or $(n-1)/2$ pairs of states. In one round, we consider all pairs of states, to see if one of their successor pairs has been found distinguishable, so a round surely takes no more than O(n^2) time. Moreover, if on some round, no additional x's are placed in the table, then the algorithm ends. Thus, there can be no more than O(n^2) rounds and O(n^4) is surely an upper bound on the running time of the table-filling algorithm.

However, a more careful algorithm can fill the table in O (n^2) time. The idea is to initialize, for each pair of states {r, s}, a list of those pairs {p, q} that "depend on" {r, s}. That is, if {r, s} is found distinguishable, then {p, q} is distinguishable. We create the lists initially by examining each pair of states {p, q}, and for each of the fixed number of input symbols a, we put {p, q} on the list for the pair of states {δ (p, a), δ (p, a)} which are the successor states for p and q on input a.

If we ever find {r, s} to be distinguishable, then we go down the list for {r, s}. For each pair on that list that is not already distinguishable, we make that pair distinguishable and we put the pair on a queue of pairs whose lists we must check similarly.

The total work of this algorithm is proportional to the sum of the lengths of the lists, since we are at all times either adding something to the lists (initialization) or examining a member of the list for the first and last time (when we go down the list for a pair that has been found distinguishable). Since the size of the input alphabet is considered a constant, each pair of states is put on O (1) lists. As there are O (n^2) pairs, the total work is O (n^2).

Minimization of DFA's

Another important consequence of the test for equivalence of states is that we can "minimize" DFA's. That is, for each DFA we can find an equivalent DFA that has as few states as any DFA accepting the same language. Moreover, except for our ability to call the states by whatever names we choose, this minimum-state DFA is unique for the language. The central idea behind the minimization of DFA's is that the notion of state equivalence lets us partition the states into blocks such that :

1. All the states in a block are equivalent.
2. No two states chosen from two different blocks are equivalent.

Example :

Consider the table of figure 2, where we determined the state equivalences and distinguishabilities for the states of figure 1. The partition of the states into equivalent blocks is ({A,E}, {B,H}, {C}, {D,F}, {G}). Notice that the three pairs of states that are equivalent are each placed in a block together, while the states that are distinguishable from all the other states are each in a block alone.

For the automaton of figure 3, the partition is ({A,C,D}, {B,E}). This example shows that we can have more than two states in a block. It may appear fortuitous that A, C, and D can all live together in a block, because every pair of them is equivalent and none of them is equivalent to any other state. However, as we shall see in the next theorem to be proved, this situation is guaranteed by our definition of "equivalence" for states.

Theorem :

The equivalence of states is transitive. That is, if in some DFA A = (Q, Σ, δ, q_0, F) We find that states p and q are equivalent and we also find that q and r are equivalent, then it must be that p and r are equivalent.

Proof:

Note that transitivity is a property we expect of any relationship called "equivalence". However, simply calling something "equivalence" doesn't make it transitive; we must prove that the name is justified.

Suppose that the pairs {p, q} and {q, r} are equivalent, but pair {p, r} is distinguishable. Then there is some input string w such that exactly one of $\hat{\delta}(p,w)$ and $\hat{\delta}(r,w)$ is an accepting state. Suppose, by symmetry, that $\hat{\delta}(p,w)$ is the accepting state.

Now consider whether $\hat{\delta}(q,w)$ is accepting or not. If it is accepting, then {q,r} is distinguishable, since $\hat{\delta}(q,w)$ is accepting and $\hat{\delta}(r,w)$ is not. If $\hat{\delta}(q,w)$ is nonaccepting, then {p, q} is distinguishable for a similar reason. We conclude by contradiction that {p, r} was not distinguishable and therefore this pair is equivalent.

We can use the Theorem above to justify the obvious algorithm for partitioning states. For each state q, construct a block that consists of q and all the states that are equivalent to q. We must show that the resulting blocks are a partition; that is, no state is in two distinct blocks.

First, observe that all states in any block are mutually equivalent. That is, if p and r are two states in the block of states equivalent to q, then p and r are equivalent to each other, by above Theorem

Suppose that there are two overlapping, but not identical blocks. That is, there is a block B that includes states p and q and another block C that includes p but not q. Since p and q are in a block together, they are equivalent. Consider how the block C was formed. If it was the block generated by p, then q would be in C*,* because those states are equivalent. Thus, it must be that there is some third state *s* that generated block C i.e., C is the set of states equivalent to s.

We know that p is equivalent to s, because p is in block C. We also know that p is equivalent to q because they are together in block B. By the transitivity of above Theorem q is equivalent to s. But then q belongs in block C, a contradiction. We conclude that equivalence of states partitions the states; that is, two states either have the same set of equivalent states (including themselves) or their equivalent states are disjoint. to conclude the above analysis :

Theorem : If we create for each state q of a DFA a block consisting of q and all the states equivalent to q, then the different blocks of states form a partition of the set of states. That is, each state is in exactly one block. All members of a block are equivalent and no pair of states chosen from different blocks are equivalent.

We are now able to state succinctly the algorithm for minimizing a DFA A = (Q, Σ, δ, q_0, F).

1. Use the table-filling algorithm to find all the pairs of equivalent states.
2. Partition the set of states Q into blocks of mutually equivalent states by the method described above.
3. Construct the minimum-state equivalent DFA B by using the blocks as its states. Let γ be the transition function of B. Suppose S is a set of equivalent states of A and a is an input symbol.

Then there must exist one block T of states such that for all states q in S, δ(q, a) is a member of block T. For if not, then input symbol a takes two states p and q of S to states in different blocks, and those are distinguishable by above Theorem. That fact lets us conclude that p and q are not equivalent and they did not both belong in S. As a consequence, we can let γ (S,a) = T. In addition :

(a) The start state of B is the block containing the start state of A.

(b) The set of accepting states of B is the set of blocks containing accepting states of A.

Note that if one state of a block is accepting, then all the states of that block must be accepting. The reason is that any accepting state is distinguishable from any nonexcepting state, so you can't have both accepting and nonaccepting states in one block of equivalent states.

Example :

Let us minimize the DFA from figure 1. We established the blocks of the state partition in Figure 5 shows the minimum-state automaton. Its five states correspond to the five blocks of equivalent states for the automaton of figure 1.

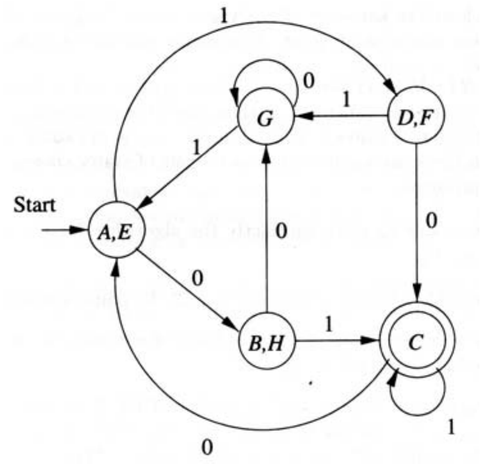


Fig. 5 : Minimum-state DFA equivalent to Fig. 1.

The start state is {A, E}, since A was the start state of figure 1. The only accepting state is {C}, since C is the only accepting states of figure 1. Notice that the transitions of figure 5 properly reflect the transitions of figure 1. For instance, figure 5 has a transition on input 0 from {A, E} to {B, H}. That makes sense, because in figure 1, A goes to B on input 0 and E goes to H. Likewise, on input 1, {A, E} goes to {D, F}. If we examine figure 1, we find that both A and E go to F on input 1, so the selection of the successor of {A, E} on input 1 is also correct. Note that the fact neither A nor E goes to D on input 1 is not important. You may check that all of the other transitions are also proper.

FINITE AUTOMATA WITH EPSILON -TRANSITIONS

We shall now introduce another extension of the finite automaton. The new "feature" is that we allow a transition on $\in$, the empty string. In effect, an NFA is allowed to make a transition spontaneously, without receiving an input symbol. This new capability does not expand the class of languages that can be accepted by finite automata, but it does not give us some added "programming convenience".

$\in$ - NFA :

We may represent an $\in$- NFA exactly as we do an NFA, with one exception : the transition function must include information about transitions on $\in$. Formally, we represent an $\in$-NFA by

N = ($Q, \Sigma, \delta, q_0, F$), where all components have their same interpretation as for an NFA, except that δ is now a function that takes as arguments :

1. A state in Q, and
2. A member of $\Sigma \cup \{ \in \}$, that is, either an input symbol or the symbol $\in$. We require that $\in$, the symbol for the empty string, cannot be a member of the alphabet Σ, so no confusion results.

Epsilon-Closure of a state

Epsilon closure of 'q' i.e. $\in$-closure (q) is denoted as the set of all states 'p' such that there is a path from 'q' to 'p' labeled '$\in$'.

Note :

- $\in$ closure (q) is the set of states with distance zero from state 'q'.
- Every state is at a distance zero to itself. It is defined as,

$$\hat{\delta}(q_0, \in) = \in\text{- closure } (q_0)$$

Conversion of NFA with $\in$ to NFA without $\in$

If NFA with $\in$-moves is given by,

$$M_1 = (Q, \Sigma, \delta, q_0, F)$$

where $\delta : Q \times (\Sigma \cup \{\in\}) \rightarrow 2^Q$

then it can be converted to NFA without $\in$-moves

$$M_2 = (Q, \Sigma, \delta', q_0, F')$$

where $\delta' : Q \times \Sigma \rightarrow 2^Q$

Set of final states F and F' may or may not be the same but Q remains the same.

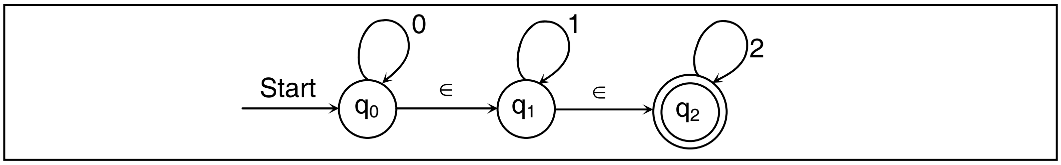
Note :
All the states whose $\in$-closure includes final states are marked as final states.

Rules for finding state function δ'

$$\delta'(q, a) = \in\text{-closure}\,(\hat{\delta}\,(q, \in), a)$$

where $\hat{\delta}(q, \in) = \in\text{-closure}\,(q)$

Convert the given NFA with $\in$ moves to NFA without $\in$ moves :



From the definition of $\in$-closure,

$\in$-closure (q_0) = $\{q_0, q_1, q_2\}$
$\in$-closure (q_1) = $\{q_1, q_2\}$
$\in$-closure (q_2) = $\{q_2\}$

$\because$ $\in$-closure of q_0 and q_1 contains q_2, the final state. Therefore, q_0 and q_1 are also final states.

$\therefore$ $F' = \{q_0, q_1, q_2\}$

Transition function δ' can be obtained with the help of given rule.

$$\begin{aligned}\delta'(q_0, 0) &= \in\text{- closure}\,(\delta(\hat{\delta}(q_0, \in), 0))\\ &= \in\text{- closure}\,(\delta(\{q_0, q_1, q_2\}, 0))\\ &= \in\text{- closure}\,(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0))\\ &= \in\text{- closure}\,(\{q_0\} \cup \phi \cup \phi)\\ &= \in\text{- closure}\,(q_0)\end{aligned}$$

$\therefore$ $\delta'(q_0, 0) = \{q_0, q_1, q_2\}$

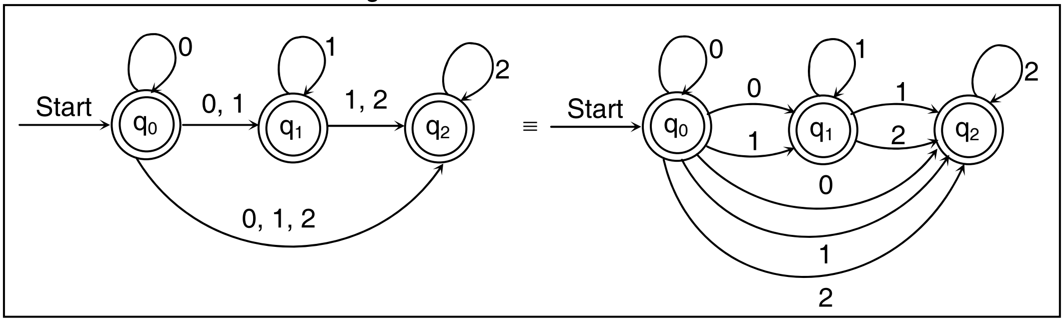
This implies that from q_0 on zero, mark transitions to q_0, q_1, q_2
Similarly,

$$\begin{aligned}\delta'(q_0, 1) &= \{q_1, q_2\}\\ \delta'(q_0, 2) &= \{q_2\}\\ \delta'(q_1, 0) &= \phi\\ \delta'(q_1, 1) &= \{q_1, q_2\}\\ \delta'(q_1, 2) &= \{q_2\}\\ \delta'(q_2, 0) &= \phi\\ \delta'(q_2, 1) &= \phi\\ \delta'(q_2, 2) &= q_2\end{aligned}$$

$\therefore$ Final T.T. is,

δ'	0	1	2
Start → q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$

∴ The resultant transition diagram is,



Note : The above NFA is for $0^* 1^* 2^*$

FINITE AUTOMATA WITH OUTPUT

When it is required to give the output, which is not just 'accept' or 'reject', it will be required to associate the required output along with each input dependent on the state of machine. Such an output can be given during the transition i.e. in a particular state q_i on receiving input i we may switch to state q_j and output is symbol 'x'.

Or only the transition is carried out and whenever a state is reached, that state will result in some output. The required output could be during transition or on reaching a state.

There are two such machines which result in output string in the two different ways stated above.

1. **Moore Machine :**
   It is the machine with finite number of states and for which, the output symbol depends upon the present state of machine.

Definition :
A Moore machine is a six-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$
where Q : finite set of states (i.e. the input alphabet)
Σ : finite set of inputs (i.e. the output alphabet)
Δ : finite set of outputs
δ : state function, $\delta : Q \times \Sigma \to Q$
λ : machine/ output function, $\lambda : Q \to \Delta$
q_0: initial state

Thus, for Moore machine, an output symbol is associated with each state. When the machine is in a particular state, it produces the output, irrespective of the input on which the transition is made.

Note :
There is nothing like accepting a string. E for every input character, only an output string or output character will be generated.

2. **Mealy Machine :**

It is the machine with a finite number of states and for which, the output symbol at a given time is a function of (i.e. depends on) the present input symbol as well as the present state of the machine.

Definition :
A Mealy machine is a six-tuple
$(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

where	Q :	finite set of states
	Σ :	finite set of inputs
	Δ :	finite set of outputs
	δ :	state function, $\delta : Q \times \Sigma \rightarrow Q$
	λ :	machine/ output function, $\lambda : Q \times \Sigma \rightarrow \Delta$
	q_0:	initial state

Thus, for this type of machine output depends on both current state and the current input symbol.

Example : Design a Moore machine to get the 1's complement of a given binary string.

Solution:

As we begin the design, we find that there are 2 different output symbols, and the number of states required will be a minimum of 2. In 1's complement 0's will be changed to 1, and 1's will be changed to 0's.

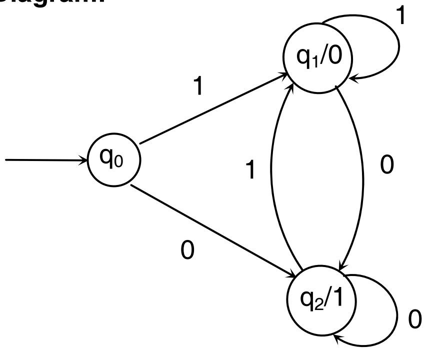
Hence there will be state giving output 0 and will be reached on receiving 1 in any state. Let q_0 be the start state and as initially we are at q_0 the output is not observed.

State q_1 gives output 0 and will be reached from q_0, q_1, q_2 when 1 is received. State q_2 gives output 1 when 0 is received in any state.

State Table :

	0	1	o/p
q_0	q_2	q_1	1
q_1	q_2	q_1	0
q_2	q_2	q_1	1

Transition Diagram:



Equivalence of Moore and Mealy Machines

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

where, $\lambda'(q, a) = \lambda(\delta(q, a))$

Illustration

Construct Moore and equivalent Mealy machine for printing the residue modulo 5 of the input treated as a ternary (base 3) number

$\Sigma = \{0, 1, 2\}$

Ternary number :

i) If ‘i’ is a ternary number, and if we write ‘0’ after it, then its value becomes ‘3i’
ii) And if we write ‘1’ after ‘i’, then its value becomes ‘3i + 1’
iii) And if we write ‘2’ after ‘i’, then its value becomes ‘3i + 2’

As, we are dividing by 5, we can have 5 different residues viz, 0, 1, 2, 3, 4. Therefore, for the 5 outputs, we create 5 different states plus one for initial state.

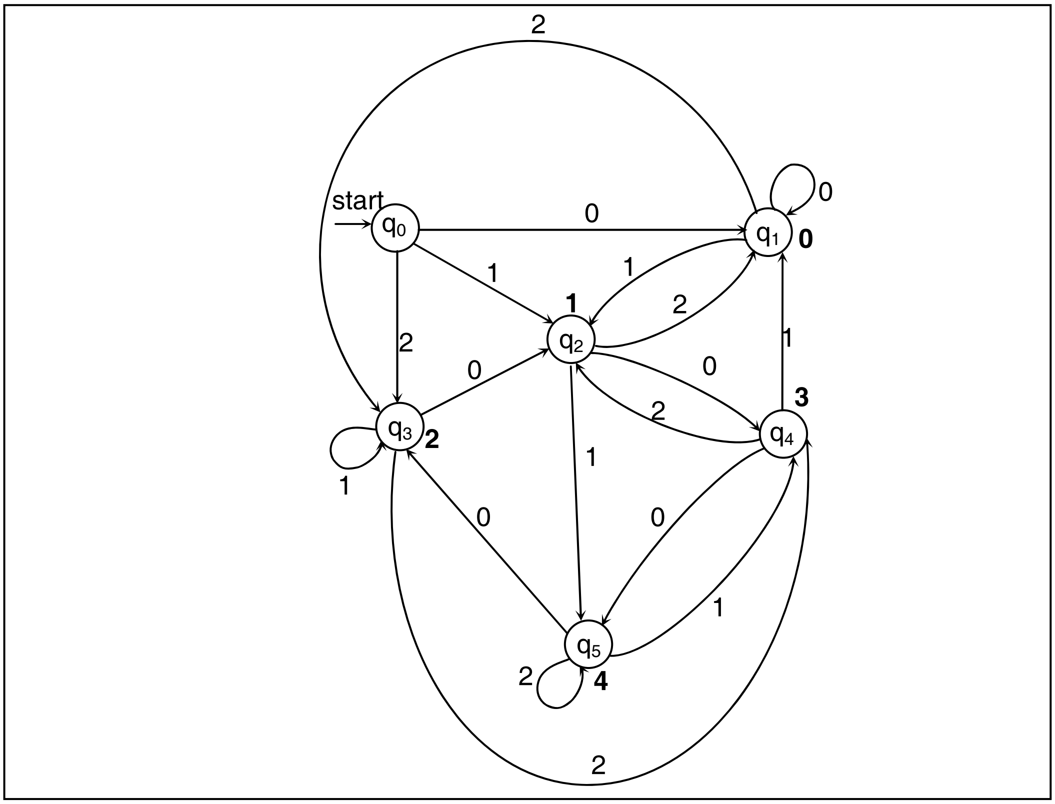
∴ First we will construct a Moore machine.

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$
$\Sigma = \{0, 1, 2\}$
$\Delta = \{0, 1, 2, 3, 4\}$
$q_0 =$ initial state
δ and λ is as shown below.

Transition Table :

	δ	0	1	2	λ(o/p)
Remainders	start q_0	q_1	q_2	q_3	–
0	q_1	q_1	q_2	q_3	0
1	q_2	q_4	q_5	q_1	1
2	q_3	q_2	q_3	q_4	2
3	q_4	q_5	q_1	q_2	3
4	q_5	q_3	q_4	q_5	4

Transition Diagram :



Equivalent Mealy :

$$\delta : Q \times \Sigma \rightarrow Q$$

$$\lambda'(q, a) = \lambda(\delta(q, a))$$

$$\lambda' : Q \times \Sigma \rightarrow \Delta$$

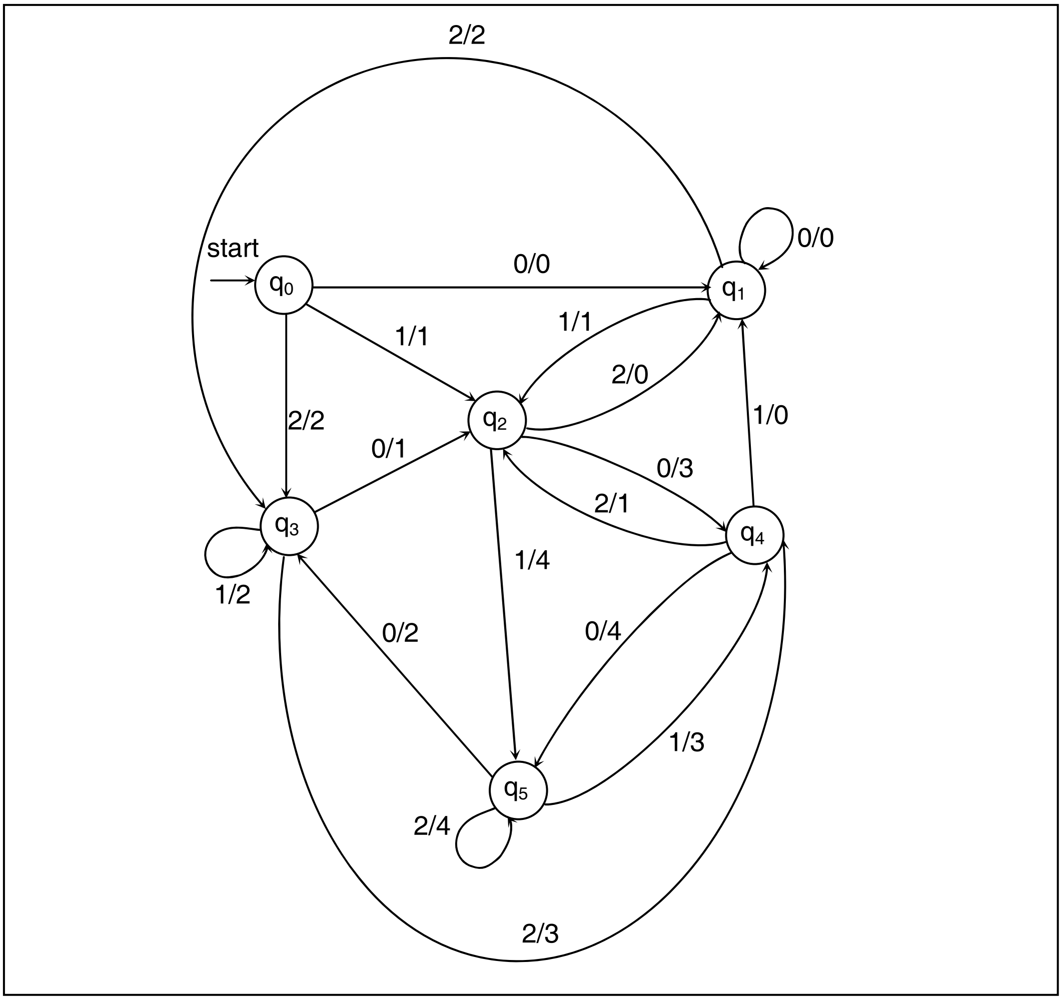
Machine/output table :

i/ps

λ	0	1	2
q_0	0	1	2
q_1	0	1	2
q_2	3	4	0
q_3	1	2	3
q_4	4	0	1
q_5	2	3	4

o/ps

Transition Diagram :

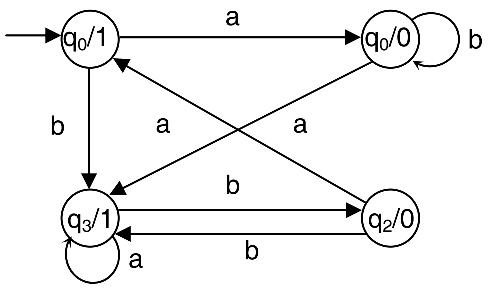


Convert the Moore machine into a Mealy machine.

	q	b	o/p
q_0	q_1	q_3	1
q_1	q_3	q_1	0
q_2	q_0	q_3	0
q_3	q_3	q_2	1

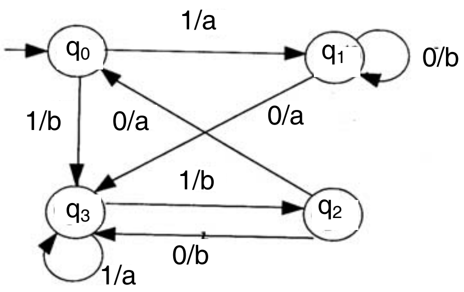
Solution:

To convert a Moore machine to a Mealy Machine the following procedure will be followed:
First draw the given Moore machine as a transition diagram.



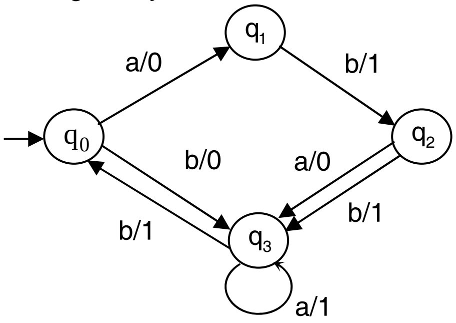
Now the output given by each state will be attached along with input symbol on all the edges which are directed towards that state.

Hence we get transition diagram as given below, which will be the required Mealy machine.



The fact is to be remembered is that, If Moore and Mealy machines are compared with each other, the number of states of the corresponding Mealy machine will be always less that the corresponding Moore machine. (At the most they could be equal.) This means that from Moore to Mealy conversion we should try to minimize the states. In this case it is not possible to merge the states as the Moore machine does not have identical rows.

: Convert the following Mealy machine to Moore machine.

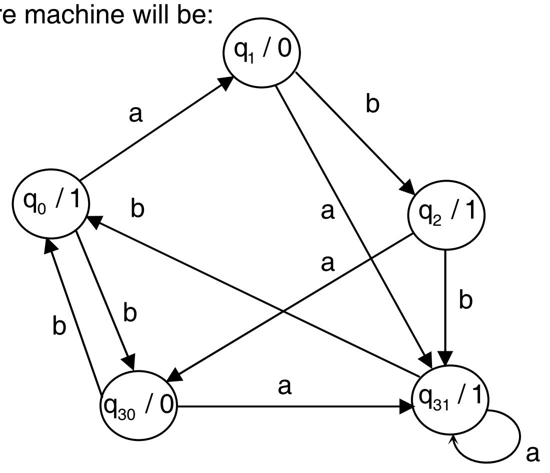


Solution:

Now as we are aware of the fact that in a Moore machine the states are associated with the output we could follow a procedure that is exactly the reverse of the previous problem, provided that all the edges coming to a state have the same output symbol. In that case the output could be directly attached to the state.

Observe that q_0 has only one incoming edge with output 1. Hence the output for q_0 state is 1. Similarly the output of state q_1 is 0 and that of q_2 is 1. But q_3 has some incoming edges with output 0 and some with output 1. Hence during the design of Moore machine the state q_3 should be divided into two states, which respectively result in output 0 and 1. The edges which are going out from state q_3 in the given diagram will now be going from each of q_{30} and q_{31}.

The required Moore machine will be:



The self loop at q_3 is now with q_{31} it is producing output 1, and there will be an edge from q_{30} and q_{31} on input 'a' which is the result of self loop at q_3, imposed on q_{30}.

The state split up will be the main factor in the conversion of Mealy to Moore. The states are also associated with the corresponding output symbols, and hence due to different outputs different states will be generated. This will again indicate corresponding Moore machine is likely to have more states that the Mealy machine. The answer should be written in the form of the transition table along with the output table, in case of Moore machine.

Distinguishing one string from another

Using a finite automaton to recognize a language L depends on the fact that there are groups of strings so that strings within the same group do not need to be distinguished from each other by the machine. In other words, it is not necessary for the machine to remember exactly which string within the group it has read so far; remembering which group the string belongs to is enough. The number of distinct states the FA need in order to recognize a language is related to the number of distinct strings that must be distinguished from each other. The following definition specifies precisely the circumstances under which an FA recognizing L must distinguish between two strings x and y, and the lemma that follows spells out explicitly how such an FA accomplishes this. (It says simply that the FA is in different states as a result of processing the two strings).

Definition :

Let L be a language in Σ^*. The set L/x is defined as follows :

$$L/x = \{z \in \Sigma^* \mid xz \in L\}$$

Two strings x and y are said to be *distinguishable* with respect to L if ; $L/x \neq L/y$. Any string z that is in one of the two sets but not the other (i.e., for which $xz \in L$ and $yz \notin L$, or vice versa) is said to distinguish x and y with respect to L. If L/x = L/y, x and y are indistinguishable with respect to L.

In order to show that two strings x and y are distinguishable with respect to a language L, it is sufficient to find one string z so that either $xz \in L$ and $yz \notin L$, or $xz \notin L$ and $yz \in L$ (in other words, so that z is one of the two sets L/x and L/y but not the other).

Properties of Finite State Machine

1. **Periodicity :**
   FSM does not have capacity to remember arbitrarily long amount of information because it has finite number of states. Hence it will repeat those states again and again i.e. some sequence of states will be repeated periodically.
   This is very important observation which will take us to the pumping lemma.

2. **Equivalent Classes of Sequences :**
   Due to periodicity, FSM partitions set of input sequences into equivalence classes.
   The equivalence class or two sequences are said to be equivalent if both of them when received in state q_i take us to the state q_j.
   Suppose I_1 & I_2 are two strings and we find that
   $d(q_i, I_1) = q_j$ and $d(q_i, I_2) = q_j$
   then I_1 and I_2 are equivalent.

3. **State Determination :**
   When the initial state and input sequence is known, we can find the current state of the FSM. Suppose at q_0, input is 110001001111 we reach the state q_3.
   If you know the initial state and you have an input string you can reach a final state but not vice versa.

4. **Impossibility of Multiplication :**
   Multiplication of long numbers cannot be carried out in FSM, since the full length sequences are to be remembered. Depending on the size of the numbers involved a large amount of information is to be remembered for taking the 'partial sum' during multiplication and hence NO FSM can multiply arbitrarily long binary or decimal numbers.

5. **Impossibility of Palindrome Recognization :**
   FSM does not have memory. It can not recognize the centre. Hence it is impossible to recognize a palindrome.

6. **Parenthesis, Tree Representation and Well-Formedness :**
   For all opening brackets there should be closing brackets and this is impossible to remember as there is no memory.
   FSM cannot check well-formedness because we will have to remember the previous opening brackets. With the above limitations, to solve complicated problems, we should have more capable machine having finite states but also the memory.

LMR (LAST MINUTE REVISION)

- **Symbol :**
  Symbol is an abstract or a user defined entity. It cannot be formally defined.

- **Set :**
  It is a finite set of similar objects.

- **Alphabets :**
  An alphabet is a finite, nonempty set of symbols.

- **Relation :**
  It is a set of ordered pairs. The first component of the pair is from the set called 'domain' and second component is from the set called 'range'.

- **Equivalence Relation :**
  If a relation is reflexive, transitive as well as symmetric it is said to be 'Equivalence Relation'.

- **String :**
  A string over an alphabet Σ is a finite sequence of symbols chosen from alphabet Σ.

- **Empty string :**
  The empty string is the string with zero occurrences of symbols.

- **Language :**
  A language is defined as a set of valid strings for some alphabet Σ.

- **Deterministic Finite Automata :**
  A DFA has a finite set of states and a finite set of input symbols. One state is designated the start state, and zero or more states are accepting state. A transition function determines how the state changes each time an input symbol is processed.

- **Notations for DFA**
  Specifying a DFA as a five-tuple with detailed description of the δ–transition function is both tedious and hard to read. There are two preferred notations for describing automata
  1. A transition diagram, which is a graph.
  2. A transition table, which is a tabular listing of the δ function

- **Language of DFA**
  The language is denoted L(A), and is defined by
  $$L(A) = \{w \mid \hat{\delta}(q_0, w) \text{ is in } F\}$$
  That is, if L is L(A) for some DFA A, then we say L is 'regular language'.

- **Non-deterministic Finite Automaton**
  The NFA differs from the DFA in that the NFA can have any number of transitions (including zero) to next states from a given state on a given input symbol.

  A non–deterministic finite automaton, abbreviated NFA, is a 5–tuple A = $\{Q, \Sigma, \delta, q_0, F\}$ where Q and Σ are non empty finite sets, $q_0 \in Q$, $A \subseteq Q$, and $\delta : Q \times \Sigma \rightarrow 2^Q$
  Q is the set of states, Σ is the alphabet, q_0 is the initial state, and A is the set of accepting states.

- **Equivalence of DFA and NFA :**
  A string w is accepted by an NFA means that there is a sequence of moves it can make, starting in its initial state and processing the symbols of w, that will lead to an accepting state.
  NFA and DFA are equivalent to each other. In other words, for every NFA there exists an equivalent DFA accepting the same set of words. Therefore, the capabilities of NFA and its DFA are same.

- There are two machines which result in output strings:

  Moore Machine :

  It is the machine with finite number of states and for which, the output symbol depends upon the present state of machine.

  Mealy Machine :

  It is the machine with finite number of states and for which, the output symbol at a given time is a function of (i.e. depends on) the present input symbol as well as the present state of the machine.

- **Equivalence of Moore and Mealy Machines**

  If $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ is Moore machine, then equivalent Mealy machine is,
  $M_2 = (Q, \Sigma, \Delta, \delta, \lambda', q_0)$
  where, $\lambda'(q, a) = \lambda(\delta(q, a))$

❑ ❑ ❑ ❑ ❑ ❑

ASSIGNMENT – 1

Duration : 45 Min.

Max. Marks : 30

Q1 to Q7 carry one mark each

1. The language accepted by Finite Automata is,
   (A) Regular languages (B) Non–regular languages
   (C) Strings (D) Binary strings

2. Consider the transition state given below :

δ	a	b
start	q_2	q_1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

   The given table is for the FA that accepts strings having even number of a's and odd number of b's.

   The final state for the given FA is
   (A) q_1 (B) q_2
   (C) q_3 (D) Can't be determined

3. Select the most appropriate combination:
   (A) DFA, NFA, Moore (B) DFA, NFA, Mealy
   (C) DFA, Moore, Mealy (D) NFA, Moore, Mealy

4. Consider the following construction of a FA that accepts a string ending with aab for Σ = (a, b).

δ	a	B
start	q_0	q_1
q_0	q_2	q_1
q_1	q_0	q_1
q_2	q_2	q_3
$\circledcirc q_3$	q_0	q_1

   where q_3 is the final state

   Which two states in combination denotes all string ending in b ?
   (A) q_0, q_1 (B) q_1, q_3
   (C) Only q_1 is sufficient (D) Cannot be determine

5. Which of the following is true for an NFA?
   I) Final state is a set of states
   II) Q is a set of subsets of states
   III) Transition table cannot be constructed

   (A) Only II (B) Only II and III
   (C) Only I and II (D) Only I

6. Let Σ = {a, b}. Then an automaton A accepting only those words from Σ having an even number of a's requires ______ states including start state.
   (A) 2 (B) 3
   (C) 4 (D) 5

Q7 to Q18 carry two marks each

Directions for Q7 – Q8

Consider the following construction of a FA that accepts a string ending with aab for Σ = (a, b).

δ	a	B
start	q_0	q_1
q_0	q_2	q_1
q_1	q_0	q_1
q_2	q_2	Q_3
$\circledcirc q_3$	q_0	Q_1

where q_3 is the final state

7. What will be the new entries for rows q_2 and q_3 if we want the given automaton to accept strings containing substring aab ?
   (A) No changes required
   (B) New entries in q_2 and q_3 will depend upon the entries for previous states also.
   (C) $q_2 \equiv (q_2 / a,\ q_1 / b)$
   $q_3 \equiv (q_3 / a,\ q_3 / b)$
   (D) $q_2 \equiv (q_2 / a,\ q_3 / b)$
   $q_3 \equiv (q_3 / a,\ q_3 / b)$

8. Which state denotes the string ending in aa
   (A) q_1 (B) q_2
   (C) q_3 (D) cannot be determined

9. Which of the following represents a Moore machine to determine the residue mod 3 for any binary number i.e. $\Sigma = \{0, 1\}$, $\Delta = \{0, 1, 2\}$

(A)

δ	0	1	o/p
start	q_0	q_1	–
Q_0	q_2	q_1	0
Q_1	q_2	q_0	1
Q_2	q_1	q_2	2

(B)

δ	0	1	o/p
start	q_0	q_1	–
q_0	q_1	q_2	0
q_1	q_2	q_0	1
q_2	q_1	q_2	2

(C)

δ	0	1	o/p
start	q_0	q_1	–
Q_0	q_1	q_2	0
Q_1	q_2	q_0	1
Q_2	q_1	q_2	2

(D) None of these

Directions for Q. 10 – Q. 11

Consider the following transition table for a FA that accepts strings containing substring aba.

δ	a	b
start	q_0	q_1
Q_0	q_0	q_2
Q_1	q_0	A
Q_2	q_3	q_1
$\circled{q_3}$	q_3	q_3

10. What is the state that A represents?

(A) q_0 (B) q_1
(C) q_2 (D) q_3

11. If we replace A by q_0, which of the following strings will be accepted ?

(A) bbbbba (B) abbbbba
(C) bbbbbba (D) abbbbbba

12. Consider the transition table given below

δ	a	B
Start	q_2	q_1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

The given table is for the FA that accepts strings having even number of a's and odd number of b's.
Which states denotes :
i) odd number of a's and even number of b's
ii) odd number of a's and odd number of b's

(A) q_0, q_1 (B) q_0, q_2
(C) q_2, q_3 (D) q_3, q_2

13. Using the given information only which is the best option that could denote Epsilon–closure of q_0

Given : 1. $Q = \{q_0, q_1, q_2, q_3, q_4\}$

2. $\in$-closure (q_2) includes q_4

(A) $\{q_0, q_1, q_2\}$ (B) $\{q_1, q_2, q_3\}$

(C) $\{q_0, q_2, q_4, q_5\}$ (D) $\{q_0, q_2, q_4\}$

14. Match the following :

(A)		(B)	
1	The set of all strings over {0, 1} having atmost one pair of 1's	i.	b* + (b + abb)* ab*
2	The set of all strings over {a, b} in which there are at least two occurrences of b between any two occurrences of a.	ii.	(1 + 01)* + (1 + 01)* 00(1 + 01)* + (0 + 10)* + (0 + 10)* 11(0 + 10)*
3	The set of all strings over {0, 1} beginning with 00.	iii.	00 (0 + 1)*

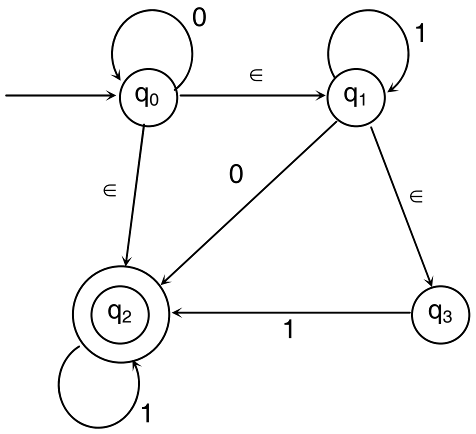
(A) 1 – ii, 2 – i, 3 – iii

(B) 1 – i, 2 – ii, 3 – iii

(C) 1 – iii, 2 – ii, 3 – i

(D) None of the above

15. After converting following NFA into DFA, the states in the final DFA will be



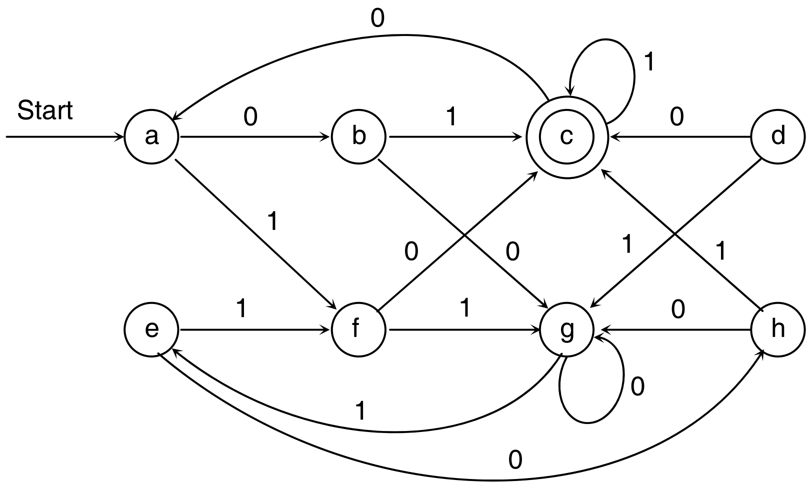
(A) $[q_1, q_2, q_3, q_0], [q_2], [q_1, q_2, q_3]$

(B) $[q_1, q_2, q_3, q_0], [q_1, q_2, q_3], [\ ]$

(C) $[q_1, q_2, q_3], [q_1, q_2, q_3, q_0], [q_0, q_2, q_3], [\ ]$

(D) $[q_1, q_2, q_3, q_0], [q_1, q_2, q_3], [q_2], [\ ]$

16. The states in equivalent minimum state DFA are



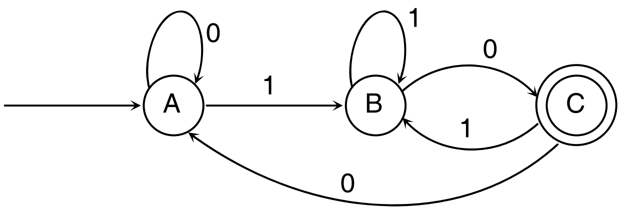
(A) $[a,e],[b,h],[c],[d,f]$

(B) $[b,h],[c],[d,f],[g],[a]$

(C) $[a,e],[b,h],[c],[d,f],[g]$

(D) $[a],[b,h],[c],[d,f],[g,e]$

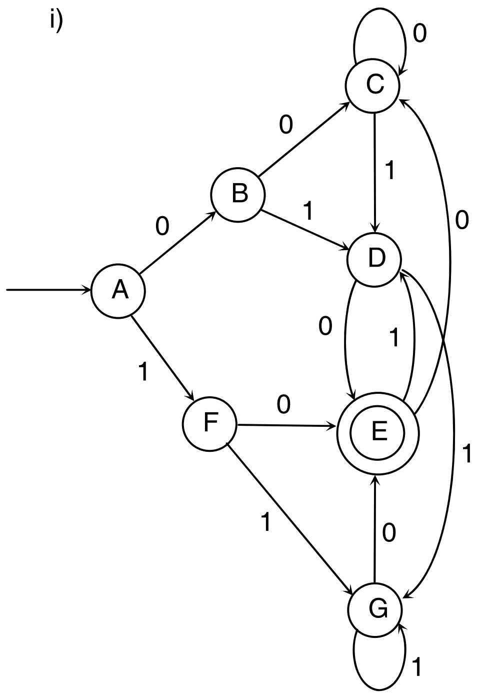
17. Following Finite Automata recognizes the language



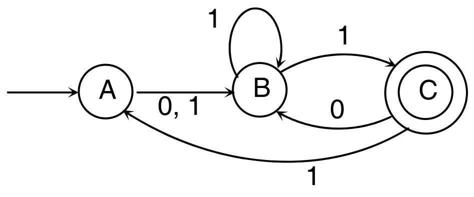
(A) 0* {10}
(B) {0, 1}* {10}
(C) 0* 1* {1, 0}
(D) 0* 11* {1, 0}

18. Which of the following FA recognizes the languages {0, 1}* {1 0}

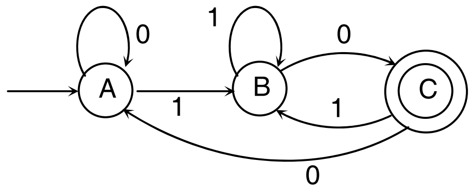
i)



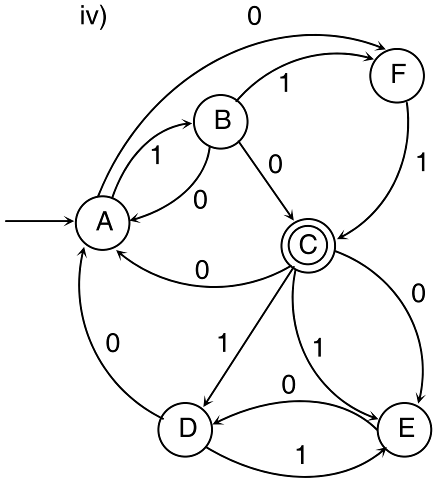
iii)



ii)



iv)



(A) i, iii
(B) ii, iii
(C) i, iv
(D) i, ii

❑ ❑ ❑ ❑ ❑ ❑

TEST PAPER – 1

Duration : 30 Min. **Max. Marks : 25**

Q1 to Q5 carry one mark each

Direction for Q1 – Q2

Consider the following transition table of an FA

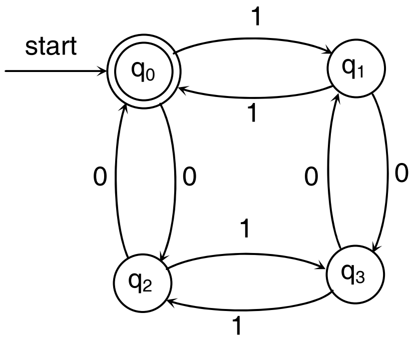
δ	a	b
start	q_1	q_0
q_0	q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_2
q_3	q_4	q_3
q_4	q_4	q_4

1. What is true of the given FA ?
   (A) Does not accept strings containing b's
   (B) Accept strings containing even no. of a's and b's
   (C) Accept strings containing odd no. of a's and even no. of b's
   (D) Accept strings independent of the no. of b's

2. If the final state is q_4, then which of the following strings will be accepted ?
   I) aaaaa
   II) aabbaabbbbb
   III) bbababbabbb

   (A) I and II (B) II and III
   (C) III and I (D) All of the above

3. Consider the transition diagram of a DFA as given below



Which is the language of the given DFA ?

(A) L = { $\in$ }
(B) L = { }
(C) L = { w | w has equal no. of ones and zeros }
(D) None of these

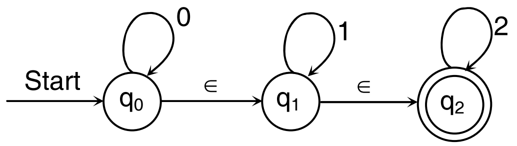
4. Consider the transition table of a DFA as given below :

δ	a	b
start	q_0	q_4
q_0	q_0	q_1
q_1	q_0	q_2
$\textcircled{q_2}$	q_0	q_2
$\textcircled{q_3}$	q_3	q_4
q_4	q_3	q_4

Which of the following is the most precise interpretation of state q_3 ?

(A) Accepts strings starting with a and ending with b
(B) Accepts strings starting with a and ending with ab
(C) Accepts strings starting with b and ending with a
(D) Accepts strings starting with a and ending with bb

5. Consider the following NFA with $\in$ moves



Which of the following strings will be accepted by the given NFA ?
I) 001122
II) 1122
III) 22

(A) Only I (B) I and II
(C) I, II and III (D) None

Q6 to Q13 carry two marks each

6. Which of the following are extended transition functions of a DFA and NFA respectively?

I) $\hat{\delta}(q, \in) = q$

$$\hat{\delta}(q, ya) = \bigcup_{r \in \hat{\delta}(q, y)} \delta(r, a)$$

II) $\hat{\delta}(q, \in) = q$

$\hat{\delta}(q, ya) = \delta(\hat{\delta}(q, y), a)$

III) $\hat{\delta}(q, \in) = \{ q \}$

$\hat{\delta}(q, ya) = \bigcup_{r \in \hat{\delta}(q, y)} \delta(r, a)$

(A) I → DFA, II → NFA

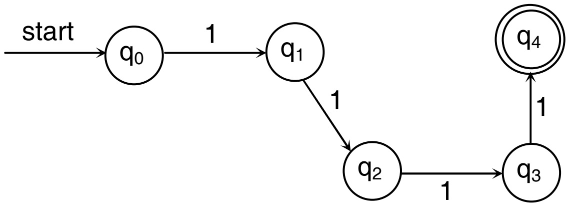
(B) I → NFA, II → DFA

(C) II → DFA, III → NFA

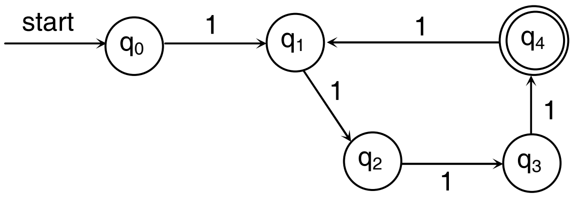
(D) II → NFA, III → DFA

7. Design a FSM to check whether a given unary number is divisible by 3

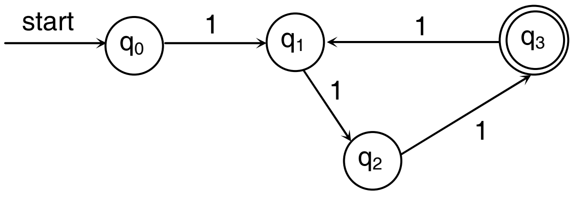
(A)



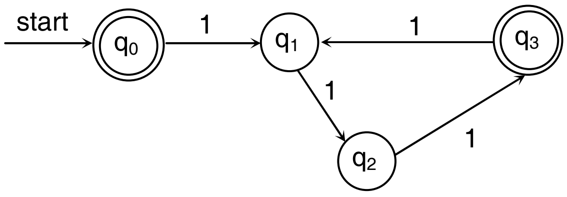
(B)



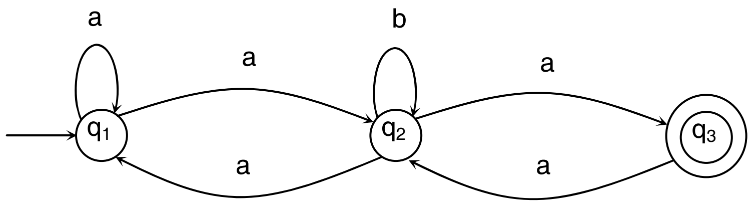
(C)



(D)



8. Consider the transition system below, find out the strings recognized by the transition system.
Transition System :



(A) $\left(a + a\,(b + a)*b + b\right)\; a\left(b + a\right)*\,a$

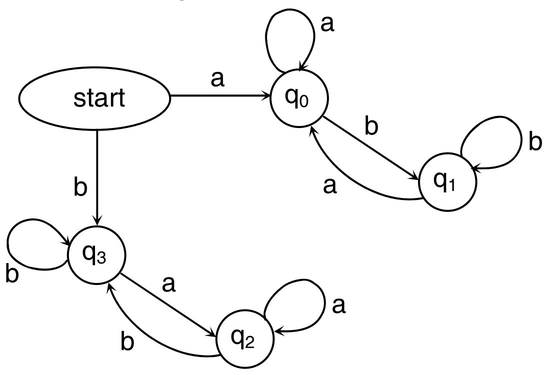
(B) $\left(a + a\,(b + aa)*b\right)\,a\left(b + aa\right)*\,a$

(C) $\left(a + a(b + aa)b\right)\,a*\left(b + a\right)\,a*$

(D) None of the above

9. Consider the transition diagram of an DFA as given below :

Which should be the final state(s) of the DFA if it should accept strings starting with 'a' and ending with 'b'.



(A) q_0

(B) q_1

(C) q_0, q_1

(D) q_3

10. The transition table given below is for the FSM that accepts a string if it ends with 'aa'.

Which is the final state ?

(A) q_0

(B) q_1

(C) q_2

(D) Can't be determined

δ	a	b
start	q_0	q_2
q_0	q_1	q_2
q_1	q_1	q_2
q_2	q_0	q_2

11. Let $\Sigma = \{0, 1\}$, then an automaton A accepting only those words from Σ having an odd number of 1's requires ______ states including the start state.

(A) 2 (B) 3
(C) 4 (D) 5

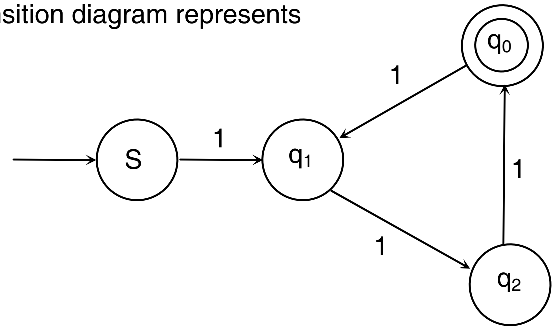
12. Consider the transition table as given below :

δ	0	1	2
$\boxed{A}$	A	B	C
$\boxed{B}$	–	B	C
$\boxed{C}$	–	–	C

If A becomes an non–accepting state, then how many strings ending with 0 will be accepted.
(Maximum length of the string is n)

(A) 0
(B) 1
(C) n
(D) n + 1

13. The transition diagram represents



(A) FSM to check whether a binary number divisible by 3
(B) FSM to check whether a given unary number is divisible by 3
(C) FSM to check whether a given unary number is divisible by 2
(D) None of the above

Q14(a) & (b) carry two marks each

Linked Answer Question

14(a). The reverse of a string can be defined more precisely by recursive rules. Which of the following statements is correct?

(A) $a^R = a^R, (wa)^R = (aw)^R$ or all $a \in \Sigma, w \in \Sigma^*$
(B) $a^R = a, (wa)^R = aw^R$ for all $a \in \Sigma, w \in \Sigma^*$
(C) $a^R = a^R, (wa)^R = a^R w^R$ for all $a \in \Sigma, w \in \Sigma^*$
(D) None of these

14(b). By using the above result, which of the following statements is correct ?

(A) $(uv)^R = v^R u^R$ for all $u, v \in \Sigma^+$
(B) $(uv)^R = vu^R$ for all $u, v \in \Sigma^*$
(C) $(uv)^R = v^R u$ for all $u, v \in \Sigma^*$
(D) None of these

❑ ❑ ❑ ❑ ❑ ❑