1.10 Conversion of NFA with Epsilon Moves to DFA
..... **Oct.-16, Dec.-18,**
..... **May-09,12,** ..... Marks 8
1.11 Difference between NFA and DFA ..... **May-09,10, Aug.-17,** .... Marks 2
1.12 Minimization of FA ..... **Dec.-12,** ..... Marks 8
1.13 Equivalence of FAs
1.14 Applications of FA
1.15 Moore and Mealy Machines **Dec.-07,11,13,16,17,**
..... **May-06,10,11, Aug.-17,**
..... **Oct.-16,18,** ..... Marks 6
1.16 Conversion of Moore to Mealy Machine ..... **May-11, Dec.-14,** ..... Marks 6
1.17 Conversion of Mealy to Moore Machine ..... **May-13,18, Dec.-16,** ... Marks 10
1.18 Fill in the Blanks
1.19 Multiple Choice Questions

Part I : Basic Concepts of Finite Automata

1.1 Basic Concepts

SPPU : May-09, Marks 4

- Automata theory deals with the definitions and properties of mathematical models of computation. These models play a role in several applied areas of computer science.
- There are two popularly used models in automata theory - **finite automaton** and **context free grammars**.
- The **finite automaton** is used in text processing, compilers and so on.
- The **context free grammars** are used in programming languages, XML and artificial intelligence.
- The automata theory helps us to begin the study of theory of computation.

Definition of automaton : An automaton is defined as a system where information is transformed and transmitted and is used for performing some functions without involvement of man.

Examples : Automatic printing machines, automatic washing machines.

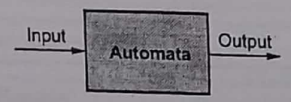


Fig. 1.1.1 Basic model of automata

The **characteristics** are -

1. **Input :** There can be finite number of inputs to the automata. These values are from the input set denoted by Σ.
2. **Output :** There can be finite number of outputs from the automata.
3. **States :** At any instance of time the automata can be in one particular state.
4. **State function :** This function is useful in determining the next state at any instance of time with the help of present state and present input.
5. **Output function :** This function is useful in determining the output at any instance of time with the help of present input and present state.

Example 1.1.1 *What is the basic machine ? Enlist the important features of basic machine.*

SPPU : May-09, Marks 4

Scanned with OKEN Scanner

Solution :

Basic machine : The basic machine is a system which models the given problem and it halts on obtaining solution to the given problem.

Important features of basic machine

1. There should be definite number of inputs to the basic machine.
2. The machine must produce proper output.
3. The performance of machine should be optimum.
4. The machine must be simple to design and implement.
5. The machine must halt after sufficient number of steps. That means it should not loop for infinitely long period.

Part II : Finite Automata

1.2 Formal Definition and Notations for FSM

A finite automata is a collection of 5-tuple $(Q, \Sigma, \delta, q_0, F)$

where,

Q is a finite **set of states**, which is non empty.

Σ is **input** alphabet, indicates input set.

q_0 is an **initial state** and q_0 is in Q i.e. $q_0 \in Q$.

F is a set of **final states.**

δ is a transition function or a mapping function. Using this function the next state can be determined.

1.2.1 Control of FA over String

The finite automata can be represented as :

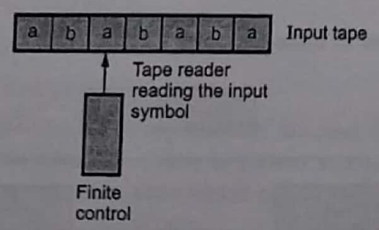


Fig. 1.2.1 Model for finite automata

The finite automata can be represented using :

i) Input tape - It is a linear tape having some number of cells. Each input symbol is placed in each cell.

ii) Finite control - The finite control decides the next state on receiving particular input from input tape. The tape reader reads the cells one by one from left to right and at a time only one input symbol is read.

For example, suppose current state is q, and suppose reader is reading the symbol 'a' then it is finite control which decides what will be the next state at input 'a'. The transition from current state q with input w to next state q′ producing w′ will be represented as,

$$(q, w) \vdash (q', w')$$

If w is a string and M is a finite automata, then w is accepted by the FA.

iff $(w, s) \vdash^* (q, \varepsilon)$

with q as final state.

The set of strings accepted by a FA given by M then M is accepted by language L. The acceptance of M by some language L is denoted by L(M).

A machine M accepts a language L iff,

$$L = L(M).$$

1.3 Concept of State Transition Diagram and Transition Table for FA

SPPU : Dec.-18, Marks 2

The strings and languages can be accepted by a finite automata, when it reaches to a final state. There are two preferred notations for describing automata :

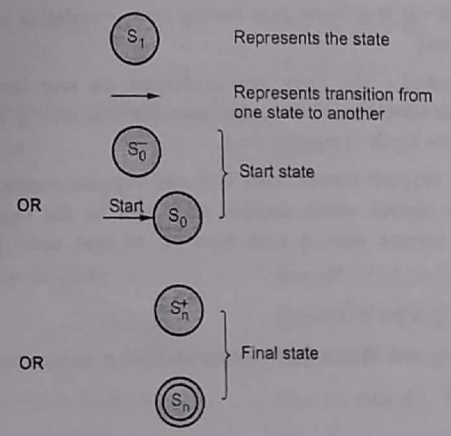
1. Transition diagram -

A transition diagram or transition graph can be defined as collection of -

1) Finite set of states K.
2) Finite set of symbols Σ.
3) A non empty set S of K. It is called **start state.**
4) A set $F \subseteq K$ of **final states.**
5) A transition function $K \times A \rightarrow K$ with K as state and A as input from Σ^*.

Scanned with OKEN Scanner

The notations used in transition diagram are -



For example :

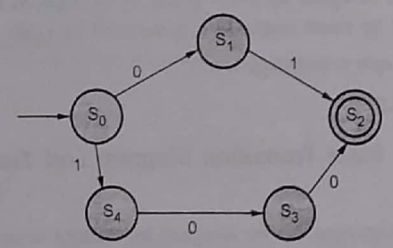
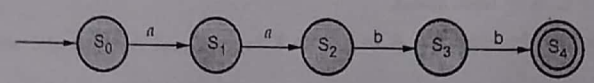


Fig. 1.3.1 Transition diagram

The FA can be represented using transition graph. The machine initially is in start state S_0 then on receiving input 0 it changes to state S_1. From S_0 on receiving input 1 the machine changes its state to S_4. The state S_2 is a final state or accept state. When we trace the input for transition diagram and reach to a final state at end of input string then it is said that the given input is **accepted** by transition diagram.

Another example :



We have drawn a transition diagram for the input **aabb**.

Note that the start state is S_0 and final state is S_4. The input set is $\Sigma = \{a, b\}$. The states S_1, S_2, S_3 are all intermediate states.

2. Transition table

This is a tabular representation of finite automata. For transition table the transition function is used.

For example :

States \ Input	a	b
q_0	q_1	-
q_1	-	q_2
q_2	q_2	-

The rows of the table corresponds to states and columns of the table corresponds to inputs.

Example 1.3.1 *State properties and limitations of FSM.* **SPPU : Dec.-18, Marks 2**

Solution : Properties :

1) FSM consists of a) States and b) State transitions.
2) An object remains in one of the state and when certain conditions are met, object changes its state.

Limitations :

1) It has no capability to remember already read input.
2) The FSM with many states and transitions can be difficult to manage.

1.4 Types of Finite Automata

- There are two types of finite automata -

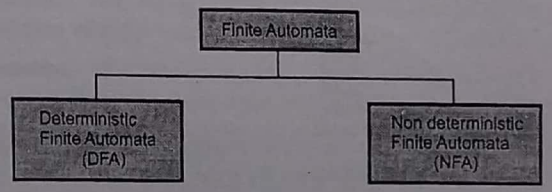


Fig. 1.4.1 DFA and NFA

Scanned with OKEN Scanner

i) Deterministic Finite Automata.

ii) Non deterministic Finite Automata.

The deterministic finite automata is deterministic in nature, in the sense, each move in this automata is uniquely determined on current input and current state. On the other hand, it is non deterministic in nature, in NFA i.e. non deterministic finite automata.

1.5 DFA

SPPU : Dec.-08,09,10,11,13,14,16,18, May-11,12,14,15,18,19, Aug.-17, Oct.-18, Marks 10

The finite automata is called Deterministic Finite Automata if there is only one path for a specific input from current state to next state. For example, the DFA can be shown as below.

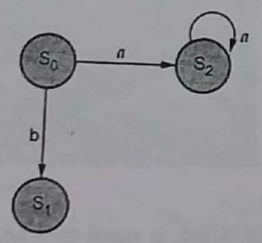


Fig. 1.5.1 Deterministic finite automata

From state S_0 for input '*a*' there is only one path, going to S_2. Similarly from S_0 there is only one path for input b going to S_1.

The DFA can be represented by the same 5-tuples described in the definition of FSM. All the above examples are actually the DFAs.

Definition of DFA

A deterministic finite automation is *a* collection of following things -

1) The finite set of states which can be denoted by Q.

2) The finite set of input symbols Σ.

3) The start state q_0 such that $q_0 \in Q$.

4) A set of final states F such that $F \in Q$.

5) The mapping function or transition function denoted by δ. Two parameters are passed to this transition function : One is current state and other is input symbol. The transition function returns a state which can be called as next state.

For example, $q_1 = \delta(q_0, a)$ means from current state q_0, with input *a* the next state transition is q_1.

In short, the DFA is a five tuple notation denoted as :

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

The name of DFA is A which is a collection of above described five elements.

Problems Based on DFA

Example 1.5.1 *Design DFA which accepts the strings that start with 1 and ends in 0.*

Solution :

- **Logic :** As the DFA contains every string which is starting with 1, there should be a transition with input 1 from start state to next state. Similarly the edge incoming to the final stage should carry 0 along with it as the strings in this DFA end with 0.
- **Design :**

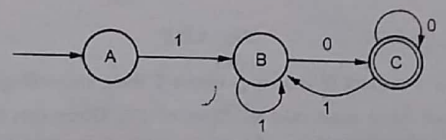


Fig. 1.5.2

- **Simulation :** Consider string 1011010. We will read each character and according move to next states.

A	⊢ 1011010
1B	⊢ 011010
10C	⊢ 11010
101B	⊢ 1010
1011B	⊢ 010
10110C	⊢ 10
101101B	⊢ 0
1011010C	⊢ ACCEPT

Thus on reading the complete string we reach to the final state. Hence it is a valid string and this string belongs to the given DFA.

Example 1.5.2 *Design FA which accepts odd number of 1's and any number of 0's.*

SPPU : Dec.-11, Marks 4

Scanned with OKEN Scanner

Solution :

- **Logic :** The odd number of 1's means the DFA can accept the 1, 111,11111,1111111,.... But there is no restriction on number of 0's i..e any number of 0's are allowed in the string.
- **Design :**

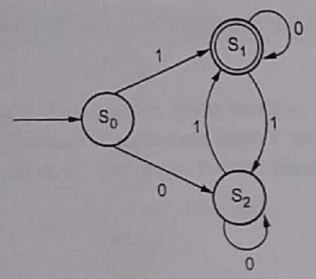


Fig. 1.5.3

- **Simulation :** At the start if we read input 1 then we will go to state S_1 which is a final state as we have read odd number of 1's. There can be any number of zeros at any state and therefore the self loop is applied to state S_2 as well as to state S_1. For example, if the input is 10101101, in this string there are any number of zeros but odd number of ones. The machine will derive this input as follows -

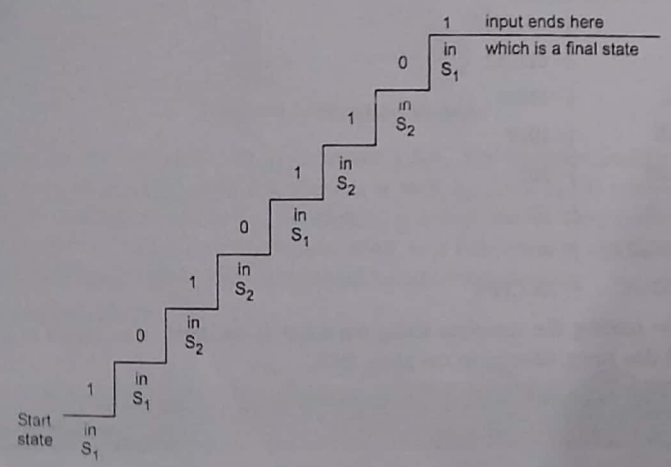


Fig. 1.5.4 Ladder diagram of processing the input

Example 1.5.3 *Design FA which checks whether the given unary number is divisible by 3.*

SPPU : Dec.-10, Marks 6

Solution :

- **Logic :** The unary number is made up of ones. The number 3 can be written in unary form as 111, number 5 can be written as 11111 and so on. The unary number which is divisible by 3 can be 111 or 111111 or 111111111 and so on.
- **Design :** The transition table is as follows -

State \ Input	1
→ q_0	q_1
q_1	q_2
q_2	q_3
q_3	q_1

- **Simulation :**

Consider a number **111111** which is equal to 6 i.e. divisible by 3. So after complete scan of this number we reach to final state q_3.

Start 111111

State q_0

$1q_1 11111$

$11q_2 1111$

$111q_3 111$

$1111q_1 11$

$11111q_2 1$

$111111q_3$ → Now we are in final state.

Example 1.5.4 *Design FA to check whether given decimal number is divisible by three.*

SPPU : May-12, Marks 8, Dec.-13, Marks 6

Solution : Logic :

- To determine whether the given decimal number is divisible by three, we need to take the input number digit by digit.
- Also, while considering its divisibility by three, we have to consider that the possible remainders could be 1, 2 or 0. The remainder 0 means, it is divisible by 3.
- Hence from input set {0, 1, 2, ..., 9} (since decimal number is a input), we will get either remainder 0 or 1 or 2 while testing its divisibility by 3.

Scanned with OKEN Scanner

- So we need to group these digits according to their remainders. The groups are as given below -

  remainder 0 group : S_0 : (0, 3, 6, 9)

  remainder 1 group : S_1 : (1, 4, 7)

  remainder 2 group : S_2 : (2, 5, 8)
- **Design :** We have named out these states as S_0, S_1 and S_2. The state S_0 will be the final state as it is remainder 0 state.

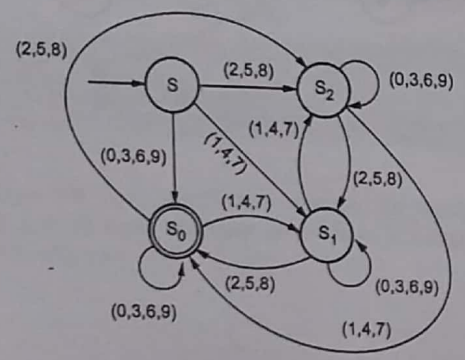


Fig. 1.5.5

- **Simulation :** Let us test the above FA, if the number is 36 then it will proceed by reading the number digit by digit.

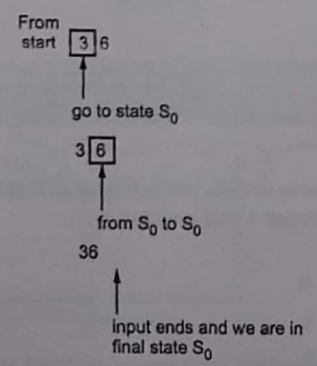


Fig. 1.5.6

Hence the number is divisible 3, isn't it ? Similarly if number is 121 which is not divisible by 3, it will be processed as

S 121

1 $\underline{S_1}$ 21

12 $\underline{S_0}$ 1

121 S_1 which is remainder 1 state.

Example 1.5.5 *Design FA which checks whether a given binary number is divisible by three.*

SPPU : Dec.-10, Oct.-18 (In Sem), Marks 6; May-18 (End Sem), Marks 4

Solution :

- **Logic :** The input number is a binary number. Hence the input set will be $\Sigma = \{0, 1\}$. The start state is denoted by S, the remainder 0 is by S_0, remainder 1 by S_1 and remainder 2 by S_2.
- **Design :**

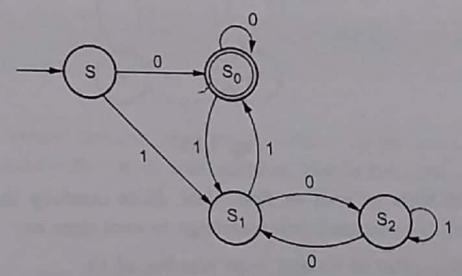


Fig. 1.5.7

- **Simulation :** Let us take some number 010 which is equivalent to 2. We will scan this number from MSB to LSB.

$$\begin{array}{ccc} 0 & 1 & 0 \\ \uparrow & \uparrow & \uparrow \\ S_0 & S_1 & S_2 \end{array}$$

Then we will reach to state S_2 which is remainder 2 state. Similarly for input 1001 which is equivalent to 9 we will be in final state after scanning the complete input.

$$\begin{array}{cccc} 1 & 0 & 0 & 1 \\ \uparrow & \uparrow & \uparrow & \uparrow \\ S_1 & S_2 & S_1 & \boxed{S_0} \end{array}$$

Thus the number is really divisible by 3.

Scanned with OKEN Scanner

Example 1.5.6 *Design FA which accepts even number of 0's and even number of 1's.*

Solution :

- Logic : This FA will consider four different stages for input 0 and 1. The stages could be

  even number of 0 and even number of 1, q_0

  even number of 0 and odd number of 1, q_1

  odd number of 0 and even number of 1, q_2

  odd number of 0 and odd number of 1. q_3
- **Design :** Let us try to design the machine

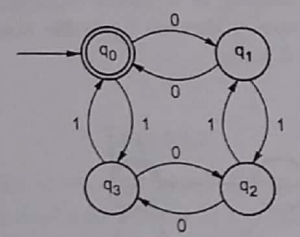


Fig. 1.5.8

Here q_0 is a start state as well as final state. Note carefully that a symmetry of 0's and 1's is maintained. We can associate meanings to each state as :

q_0 : State of even number of 0's and even number of 1's.

q_1 : State of odd number of 0's and even number of 1's.

q_2 : State of odd number of 0's and odd number of 1's.

q_3 : State of even number of 0's and odd number of 1's.

The transition table can be as follows -

State \ Input	0	1
→ q_0	q_1	q_3
q_1	q_0	q_2
q_2	q_3	q_1
q_3	q_2	q_0

Example 1.5.7 *Design FA to accept L, where L = {Strings in which a always appears trippled} over the set Σ = {a, b}.*

Solution :

- **Logic :** In this language the a always appear in a clump of three. This clump may or may not be surrounded by b.
- **Design :**

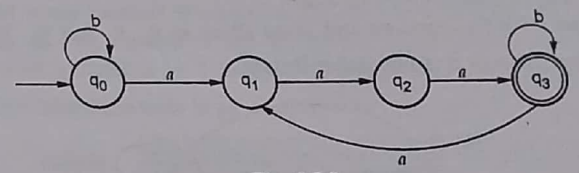


Fig. 1.5.9

- **Simulation :**

  baaabaaa

  q0 baaabaaa

  bq0 aaabaaa

  baq1 aabaaa

  baaq2 abaaa

  baaaq3 baaa

  baaabq3 aaa

  baaabaq1 aa

  baaabaaq2 a

  baaabaaaq3

Example 1.5.8 *Design FA to accept L where all the strings in L are such that total number of a's in them are divisible by 3.*

Solution :

- **Logic :** As we have seen earlier, while testing divisibility by 3, we group the input as remainder 0, remainder 1 and remainder 2.

  Hence,

  S_0 : State of remainder 0.

  S_1 : State of remainder 1.

  S_2 : State of remainder 2.

Scanned with OKEN Scanner

- **Design :**

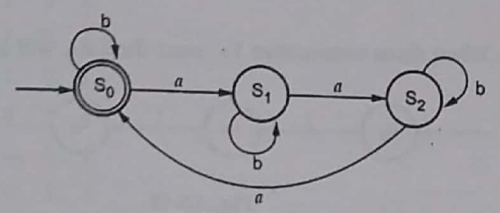


Fig. 1.5.10

Example 1.5.9 *Design a FA that reads strings made up of letters in the word. CHARIOT and recognize those strings that contain the word 'CAT' as a substring.*

SPPU : May-12, Marks 8; May-19, Marks 5

Solution :

- **Logic :** To design this FA we will first consider the input set which will be $\Sigma = \{C, H, A, R, I, O, T\}$. From this input set we have to recognize the word 'CAT'. We can do that as follows -
- **Design :**

Step 1 :

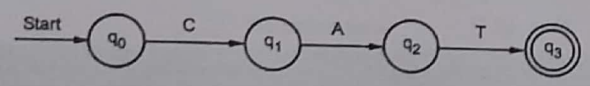


Fig. 1.5.11

Step 2 :

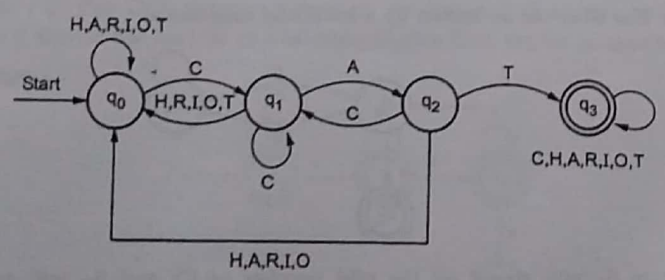


Fig. 1.5.12 Required finite automata

From Fig. 1.5.11 it is clear that on recognizing C initially from $\Sigma = \{C, H, A, R, I, O, T\}$ we are moving to state q_1, other than C all the characters are remaining in state q_0 only.

From q_1 the character A will be identified and from q_2 character T will be identifier. The transition table can be drawn as follows :

State \ Input	C	H	A	R	I	O	T
→ q_0	q_1	q_0	q_0	q_0	q_0	q_0	q_0
q_1	q_1	q_0	q_2	q_0	q_0	q_0	q_0
q_2	q_1	q_0	q_0	q_0	q_0	q_0	q_3
q_3	q_3	q_3	q_3	q_3	q_3	q_3	q_3

The substring may appear anywhere in the input string for any number of times. For example, 'HACCATHA'. From this 'CAT' can be recognized and we should reach to q_3 state finally.

Example 1.5.10 *Design DFA to accept odd and even numbers represented using binary notation.*

Solution :

- **Logic :** The binary number that ends with 0 is an even number and binary number that ends with 1 is an odd number. Hence the DFA is -
- **Design :**

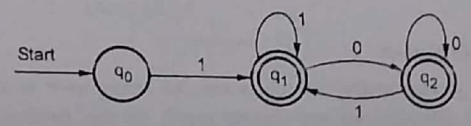


Fig. 1.5.13

Example 1.5.11 *Write a DFA to accept the language $L = \{L : |W| \bmod 5 \neq 0\}$.*

Solution :

- **Design :** The string which we obtain should not be divisible by 5. Hence the, DFA will be,

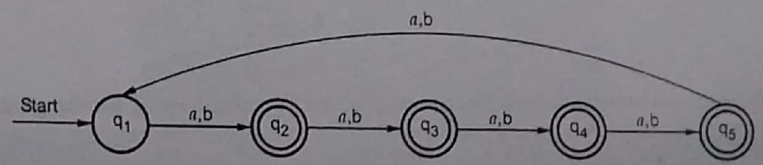


Fig. 1.5.14

Scanned with OKEN Scanner

The transition table can be drawn as follows -

State \ Input	a	b
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_1	q_1

- **Simulation :** We can consider the string "*a*bbb". This string is a valid string as it is not divisible by 5 i.e *a*bbb mod 5 ≠ 0. That also means that we should reach to final state on acceptance of this string. The simulation of this string for given DFA is

$$\begin{aligned}\delta(q_1, abbb) &\vdash \delta(q_2, bbb)\\ &\vdash \delta(q_3, bb)\\ &\vdash \delta(q_4, b)\\ &\vdash \delta(q_5, \varepsilon)\end{aligned}$$

where q_5 is a final state.

Now consider string "*ababa*" which is invalid string as it is divisible by 5. That means ababa mod 5 = 0. That means we should reach to non final state on acceptance of this string. The simulation for this string is as given below.

As q_1 is a start we will start from state q_1.

$$\begin{aligned}\delta(q_1, ababa) &\vdash \delta(q_2, baba)\\ &\vdash \delta(q_3, aba)\\ &\vdash \delta(q_4, ba)\\ &\vdash \delta(q_5, a)\\ &\vdash \delta(q_1, \varepsilon)\end{aligned}$$

Here q_1 is a non-final state. That means "*ababa*" is invalid string for the given DFA.

Example 1.5.12 *Design a DFA $L(M) = \{W \mid W \in \{0,1\}^*\}$ and W is a string that does not contain consecutive 1's.*

Solution :

- **Design :** When three consecutive 1's occur the DFA will be

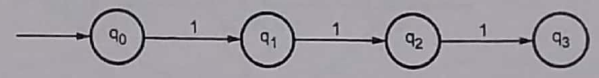


Fig. 1.5.15

Here two consecutive 1's or single 1 is acceptable, hence

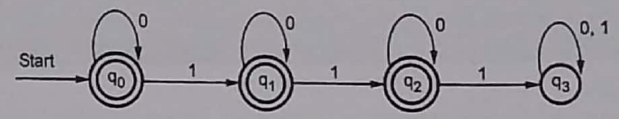


Fig. 1.5.16

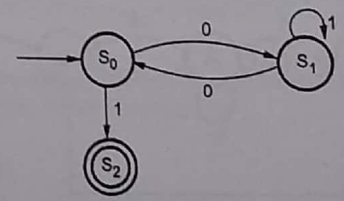
The states q_0, q_1, q_2 are final states. Hence DFA M can be represented as,

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

where
$$Q = \{q_0, q_1, q_2, q_3\}$$
$$\Sigma = \{0,1\}$$
$$F = \{q_0, q_1, q_2\}$$

Example 1.5.13 *Design a DFA which accepts strings with even number of 0's followed by single 1 over $\Sigma = \{0,1\}$.*

Solution : The DFA can be shown by a transition diagram as



Here state S_1 will accept all the odd number of 0's and S_0 will accept all even number of 0's. It should then be followed by single 1. Hence S_2 is a final state. Consider the input

Scanned with OKEN Scanner

	0	0	0	1	0	1
0	S_1	0	0	1	0	1
0	0	S_0	0	1	0	1
0	0	0	S_1	1	0	1
0	0	0	1	S_1	0	1
0	0	0	1	0	S_0	1
0	0	0	1	0	1	S_2

We now will reach to final state S_2, and the input ends here. Hence 000101 is a valid input string. The transition table for above drawn DFA can be represented as

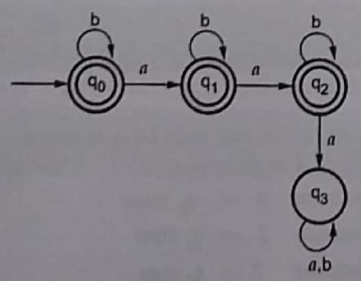
States \ Input	0	1
→ S_0	S_1	S_2
S_1	S_0	S_1
S_2	-	-

Transition table

Example 1.5.14 *Design DFA which accepts all the strings not having more than two a's over $\Sigma = \{a, b\}$.*

Solution :

- **Logic :** In this DFA maximum two a's are accepted. If we try to accept third a then it should not lead us to final state. Such a DFA can be as shown below.
- **Design :**



The transition table can be as shown -

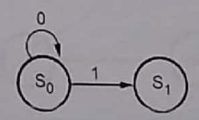
States \ Input	a	b
→ q_0	q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_2
q_3	q_3	q_3

Transition table

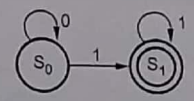
Example 1.5.15 *Design a DFA for accepting all the strings of $\{L = 0^m 1^n \mid m \geq 0 \text{ and } n \geq 1\}$.*

Solution :

- **Logic :** This is a language in which all the 0's followed by all 1's. But number of zero's and number of 1's are different. The language explicitly mentions that there should be at least one 1. Any number of 0's followed by only one 1 is -



- **Design :** But we have 1^n in the language. Hence the DFA can be



The transition table can be

State \ Input	0	1
S_0	S_0	S_1
S_1	-	S_1

Scanned with OKEN Scanner

- **Simulation :** Consider a string 00111, which is a valid string and is accepted by above drawn DFA.

$$\begin{aligned}\delta(S_0, 00111) &\vdash (S_0, 0111)\\ &\vdash (S_0, 111)\\ &\vdash (S_1, 11)\\ &\vdash (S_1, 1)\\ &\vdash (S_1, \varepsilon)\end{aligned}$$

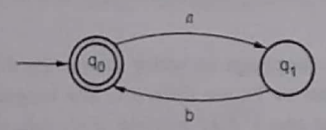
Thus we reach to final state S_1 on acceptance of 00111.

Example 1.5.16 *Design DFA over $\sum = \{a, b\}$ for*
i) $(ab)^n$ with $n \geq 0$ ii) $(ab)^n$ with $n \geq 1$.

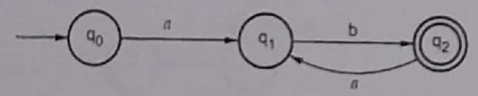
Solution :

- This is a language of ab in a pair in
  i) Condition ε is accepted and in
  ii) Condition ε is not accepted.

The DFA for i) can be



ii) The DFA will be -



The string accepted by

i) $\delta(q_0, ab) \vdash (q_1, b)$
$\vdash (q_0, \varepsilon)$

Here q_0 is a final state.

ii) $\delta(q_0, ab) \vdash (q_1, b)$
$\vdash (q_2, \varepsilon)$

Here q_2 is final state.

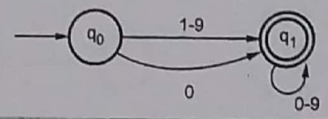
Example 1.5.17 *Design DFA for accepting the set of integers.*

Solution :

- **Logic :** For representing any integer the set {0, 1, ... 9} is used. We can represent any integer value by taking appropriate combinations of symbols from {0, ... 9}.

Hence the DFA will be

- **Design :**



Example 1.5.18 *Construct a DFA for accepting L over {0, 1} such that every substring of length 4 contains at least three 1's.*

SPPU : Dec.-08, Marks 8, Dec.-16, End Sem, Marks 4

Solution :

- **Logic :** The r.e. for designing the required DFA is

Regular expression = $(0111 + 1011 + 1101 + 1110)^*$.

Hence we can design the required DFA as follows :

- **Design :**

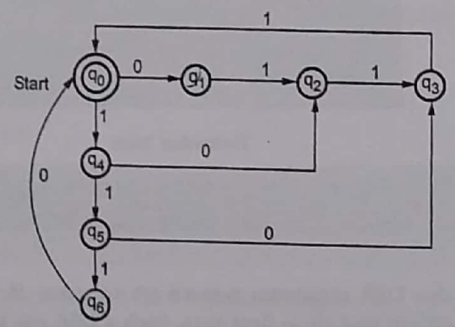


Fig. 1.5.17

Example 1.5.19 *Design a FSM for divisibility of 5 tester of a given decimal number.*

SPPU : Dec.-11, Marks 8

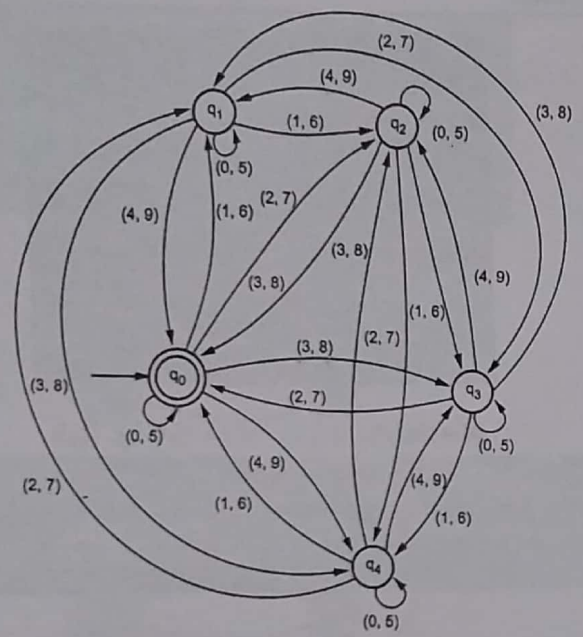
Solution :

- **Logic :** The logic for designing such DFA is simple. First of all we will group the decimal numbers 0 to 9 as follows :

(0, 5) → remainder 0 → q_0 state
(1, 6) → remainder 1 → q_1 state
(2, 7) → remainder 2 → q_2 state
(3, 8) → remainder 3 → q_3 state
(4, 9) → remainder 4 → q_4 state

Scanned with OKEN Scanner

Then the required DFA can be drawn as follows -



Now assume input number 365. The state transition will be

$\delta(q_0, 3) = q_3$

$\delta(q_3, 6) = q_4$

$\delta(q_4, 5) = q_4$

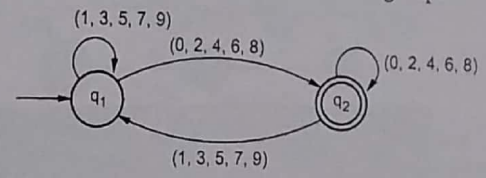
That means on accepting input 365 we will reach state q_4 which is actually a **remainder 4 state.** Similarly if we sum up the digits of number 365 then we get $3 + 6 + 5 = 14$ and $14/5$ = **remainder 4.**

If the sum of digits is 5, 10, 15 and so on then we must reach in q_0 state. The q_0 state denotes that sum of digits of a given number is divisible by 3.

Example 1.5.20 *Design FSM to check for divisibility of a decimal number by 2.*

SPPU : Dec.-09, Marks 6

Solution : If any decimal number contains 0, 2, 4, 6 or 8 at its unit place then that number is always divisible by 2. Hence we will make two groups.



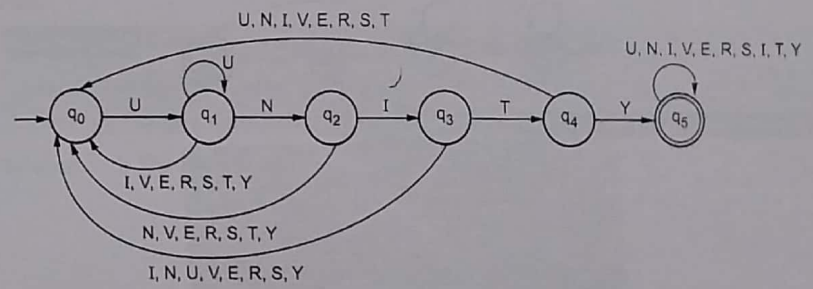
(0, 2, 4, 6, 8) → q_2 state

(1, 3, 5, 7, 9) → q_1 state

Example 1.5.21 *Design a finite automaton that reads strings made of letter in the word "UNIVERSITY" and recognize these strings that contain the word UNITY as substring.*

SPPU : Dec.-09, Marks 6

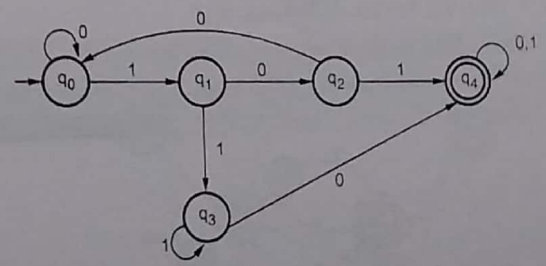
Solution : The finite automaton can be drawn as follows -



Example 1.5.22 *Design a FSM to concept those strings having 101 or 110 as substring.*

SPPU : May-11, Marks 4

Solution : The FSM will be

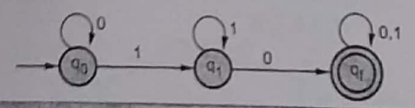


Example 1.5.23 *Design a DFA for a language of strings of 0's and 1's such that - i) Substring is 10 ii) Strings ending with 101.*

SPPU : Dec.-12, Marks 10

Scanned with OKEN Scanner

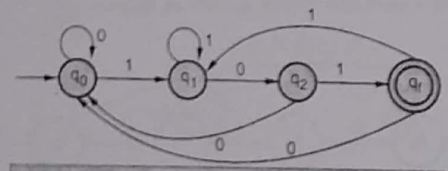
Solution : i)



Q \ Σ	0	1
→ q_0	q_0	q_1
q_1	q_f	q_1
* q_f	q_f	q_f

ii)

$M = (\{q_0, q_1, q_f\}, \{0, 1\}, \delta, q_0, \{q_f\})$



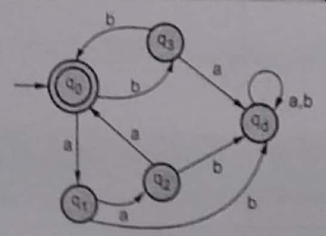
Q \ Σ	0	1
→ q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_0	q_f
* q_f	q_0	q_1

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

Example 1.5.24 *Obtain a DFA to accept strings of a's and b's such that -*
$L = \{W / W \in (a+b)^*$ such that $N_a(W) \bmod 3 = 0$ and $N_b(W) \bmod 2 = 0\}$

SPPU : Dec.-12, Marks 10

Solution :



Q \ Σ	a	b
→ * q_0	q_1	q_3
q_1	q_2	q_d
q_2	q_0	q_d
q_3	q_d	q_0
q_d	q_d	q_d

$M = (\{q_0, q_1, q_2, q_3, q_f\}, \{a, b\}, \delta, q_0, \{q_0\})$

Example 1.5.25 *Design a DFA for following language*
$L = \{W \mid W$ is Binary word of length $4i$ (where $i >= 1$) such that each consecutive block of 4 bits contains atleast 2 0s$\} = \{0000, 0110, 01101100, \ldots\}$

SPPU : Dec.-13, Marks 10; Aug.-17, Marks 8

Solution :

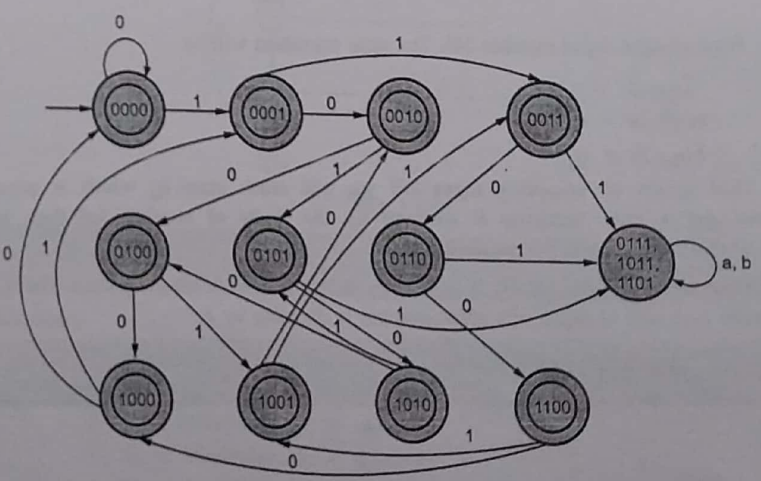


Fig. 1.5.18

Example 1.5.26 *Design a FSM to check given decimal number is divisible by 4 or not.*

SPPU : May-14, Marks 8

Scanned with OKEN Scanner

Solution : For checking divisibility of a decimal number by 4, there are following states

q_0	Remainder 0 State. It is a final state
q_1	Remainder 1 State
q_2	Remainder 2 State
q_3	Remainder 3 State

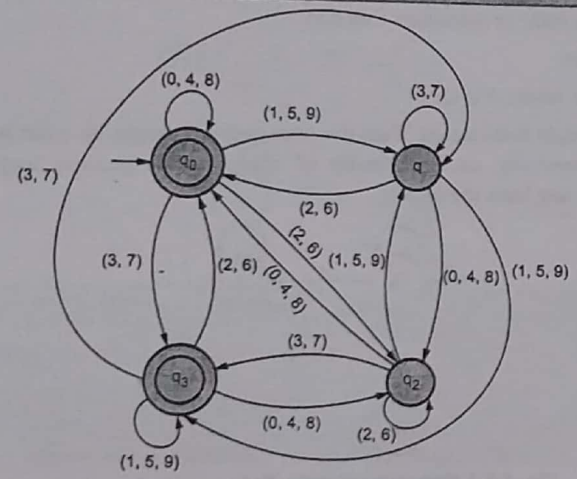


Fig. 1.5.19

Example 1.5.27 *Design finite automation for the following*

i) FA which reads strings made up of {0, 1} and accepts only those strings which end in either '00' or '11'.

ii) FA which accepts only those strings with 'a' at every even position. $\Sigma = \{a, b\}$.

SPPU : Dec.-14, Marks 6

Solution :

i)

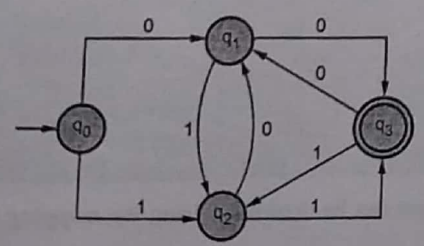


Fig. 1.5.20

ii)

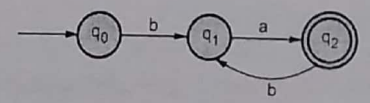


Fig. 1.5.21

Example 1.5.28 *Define Finite Automata and justify why palindrome strings cannot be checked for by FSM.*

SPPU : May-15, Marks 4

Solution : Finite Automata : Refer section 1.2.

The palindrome is a string which appears to be the same if read from left to right or from right to left. That means to recognize the palindrome, the symbols read from left to right must be remembered in order to match the symbols that can be read from right to left. To remember the symbols memory must be required in the machine. But as finite automata does not contain any memory, it is not possible to recognize palindrome using FSM.

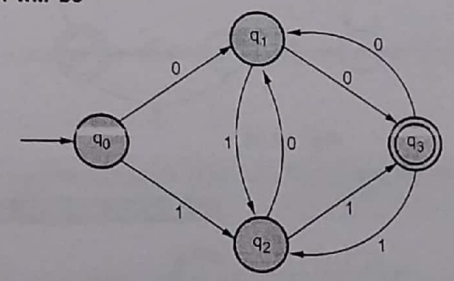
Example 1.5.29 *Design an FA over $\Sigma = \{0, 1\}$ For the following :*

i) Strings which end either "00" or "11".

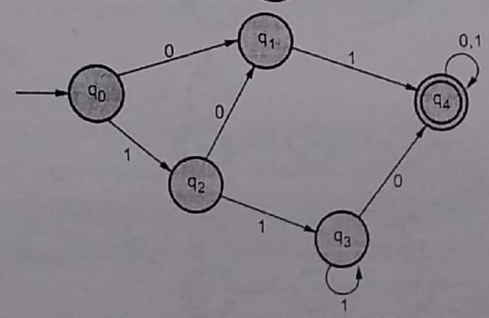
ii) Strings which contain either "01" or "110".

SPPU : Oct.-16, In Sem., Marks 5

Solution : i) The FA will be



ii)



Scanned with OKEN Scanner

Example 1.5.30 *Design FA for accepting strings over $\Sigma = \{a,b\}$.*

i) String containing at least one 'a' and at least one 'b'.

ii) Set of all strings that do not contain three or more consecutive a's

SPPU : Oct.-18 (In Sem), Marks 6

Solution : i)

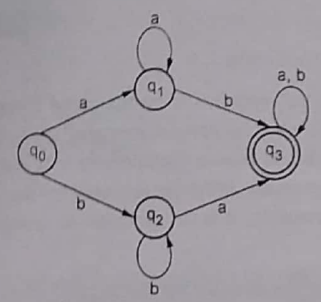


Fig. 1.5.22

ii)

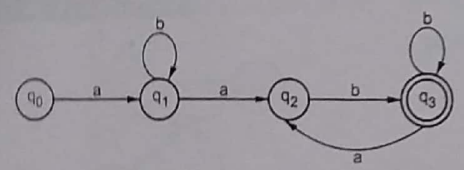


Fig. 1.5.23

Example 1.5.31 *Construct transition graph for the following regular expression.*

$r = 1^* \cdot 0 \cdot 0 \cdot (0+1)^*$

SPPU : Dec.-18 (End Sem), Marks 4

Solution : $r = 1^* \cdot 0 \cdot 0 \ (0+1)^*$

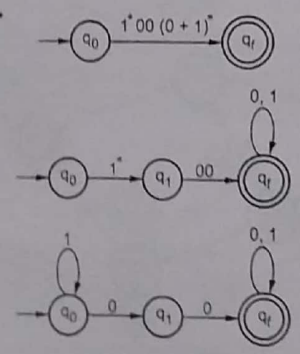


Fig. 1.5.24

1.6 NFA

Definition of NFA

The NFA can be formally defined as a collection of 5-tuples.

1) Q is a finite set of states.

2) Σ is a finite set of inputs.

3) δ is called next state or transition function.

4) q_0 is initial state.

5) F is a final state where $F \subseteq Q$.

There can be multiple final states. Thus the next question might be what is the use of NFA. The NFA is basically used in theory of computations because they are more flexible and easier to use than the DFAs.

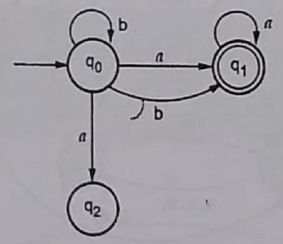


Fig. 1.6.1 Non deterministic finite automata

Problems Based on NFA

Example 1.6.1 *Design the NFA transition diagram for the transition table as given below -*

	0	1
q_0	$\{q_0, q_1\}$	$\{q_0, q_2\}$
q_1	$\{q_3\}$	
q_2	$\{q_2, q_3\}$	$\{q_3\}$
q_3	$\{q_3\}$	$\{q_3\}$

Here the NFA is $M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_3\})$.

Solution : The transition diagram can be drawn by using the mapping function as given in table.

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

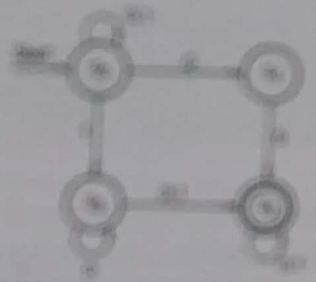
$$\delta(q_0, 1) = \{q_0, q_2\}$$

Scanned with OKEN Scanner

Then

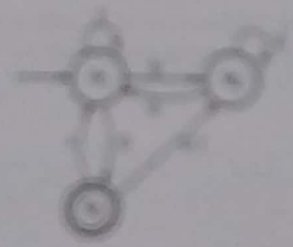
Then

Then



Solution : Finally we will design a transition table using the given mapping function.

The DFA will be



1.1.2 NFA with Epsilon Moves

The ε - transitions in NFA are given in order to move from one state to another without having any symbol from input set Σ.

Consider the NFA with ε as :

Fig. 1.1.2 : NFA with ε - transitions

Definition :

The language L accepted by NFA with ε, denoted by M = (Q, Σ, δ, q0, F) can be defined as

L(M) = {x ∈ Σ* : δ̂(q0, x) ∈ F} for a DFA with ε

where

Q is a finite set of states

Σ is input set

δ is transition or a mapping function for transitions from Q × (Σ ∪ {ε}) to 2^Q

q0 is start state

F is set of final states such that F ⊆ Q

The string x is accepted by NFA can be represented as

L(M) = {x : δ̂(q0, x) ∩ F ≠ ∅} and δ transition for a final q0 reaches to F.

Example 1.1.2 Construct DFA with ε which accepts a language consisting the string of the number divisible by any number of 0's followed by any number of 1's.

Scanned with OKEN Scanner

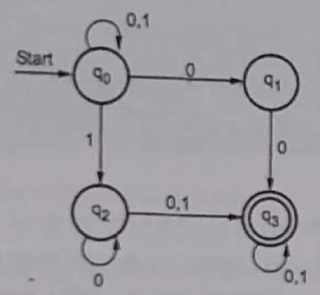
Then, $\delta(q_1, 0) = \{q_3\}$

Then, $\delta(q_2, 0) = \{q_2, q_3\}$

$\delta(q_2, 1) = \{q_3\}$

Then, $\delta(q_3, 0) = \{q_3\}$

$\delta(q_3, 1) = \{q_3\}$



Example 1.6.2 *Construct a transition diagram for the NDFA $M = (\{q_1, q_2, q_3\}, \delta, q_1, \{q_3\})$ where δ is given by,*

$\delta(q_1, 0) = \{q_2, q_3\}$ $\delta(q_1, 1) = \{q_1\}$

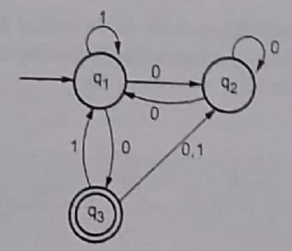
$\delta(q_2, 0) = \{q_1, q_2\}$ $\delta(q_2, 1) = \phi$

$\delta(q_3, 0) = \{q_2\}$ $\delta(q_3, 1) = \{q_1, q_2\}$

Solution : Firstly we will design a transition table using the given mapping function δ.

States \ Input	0	1
→ q_1	$\{q_2, q_3\}$	q_1
q_2	$\{q_1, q_2\}$	ϕ
q_3	q_2	$\{q_1, q_2\}$

The NDFA will be



1.7 NFA with Epsilon Moves

SPPU : Dec.-15, Oct.-16, Marks 5

The ε - transitions in NFA are given in order to move from one state to another without having any symbol from input set Σ.

Consider the NFA with ε as :

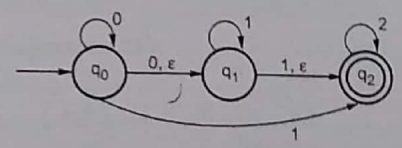


Fig. 1.7.1 NFA with ε - transitions

Definition :

The language L accepted by NFA with ε, denoted by $M = (Q, \Sigma, \delta, q_0, F)$ can be defined as :

Let, $M = (Q, \Sigma, \delta, q_0, F)$ be a NFA with ε.

where

Q is a finite set of states.

Σ is input set.

δ is a transition or a mapping function for transitions from $Q \times \{\Sigma \cup \varepsilon\}$ to 2^Q.

q_0 is a start state.

F is a set of final states such that $F \in Q$.

The string w in L accepted by NFA can be represented as

$L(M) = \{w \mid w \in \Sigma^* \text{ and } \delta \text{ transition for w from } q_0 \text{ reaches to F}\}$.

Example 1.7.1 *Construct NFA with ε which accepts a language consisting the strings of any number of a's followed by any number of b's. Followed by any number of c's.*

Scanned with OKEN Scanner

Solution :

- **Logic :** Here any number of a's or b's or c's means zero or more in number. That means there can be zero or more a's followed by zero or more b's followed by zero or more c's. Hence NFA with ε can be -
- **Design :**

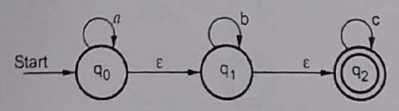


Fig. 1.7.2

Normally ε's are not shown in the input string. The transition table can be

δ \ Input (Σ)	a	b	c	ε
q_0	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_1	ϕ	q_2
q_2	ϕ	ϕ	q_2	ϕ

- **Simulation :**

We can parse the string *aa*bbcc as follows -

$\delta(q_0, aabbcc) \vdash \delta(q_0, abbcc)$

$\vdash \delta(q_0, bbcc)$

$\vdash \delta(q_0, \varepsilon bbcc)$

$\vdash \delta(q_1, bbcc)$

$\vdash \delta(q_1, bcc)$

$\vdash \delta(q_1, cc)$

$\vdash \delta(q_1, \varepsilon cc)$

$\vdash \delta(q_2, cc)$

$\vdash \delta(q_2, c)$

$\vdash \delta(q_2, \varepsilon)$

Thus we reach to accept state, after scanning the complete input string.

Definition of ε - closure

The ε - closure (p) is a set of all states which are reachable from state p on ε -transitions such that :

i) ε - closure (p) = p where $p \in Q$.

ii) If there exists ε - closure (p) = {q} and $\delta(q, \varepsilon) = r$ then ε - closure (p) = {q, r}.

Example 1.7.2 *Find ε - closure for the following NFA with ε.*

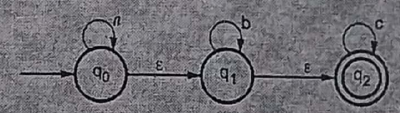


Fig. 1.7.3

Solution :

ε - closure $(q_0) = \{q_0, q_1, q_2\}$ means self state + ε - reachable states.

ε - closure $(q_1) = \{q_1, q_2\}$ means q_1 is a self state and q_2 is a state obtained from q_1 with ε input.

ε - closure $(q_2) = \{q_2\}$

University Question

1. *Write Formal definition of NFA with epsilon. Also define epsilon closure.*

SPPU : Dec.-15, End Sem, Oct.-16 In Sem, Marks 5

1.8 Conversion of NFA to DFA

SPPU : Dec.-10, May-11,16, Marks 6

The conversion method will follow following steps -

1) The start state of NFA M will be the start for DFA M′. Hence add q_0 of NFA (start state) to Q′. Then find the transitions from this start state.

2) For each state $[q_1, q_2, q_3, \ldots\ldots q_i]$ in Q′ the transitions for each input symbol Σ can be obtained as,

i) $\delta'([q_1, q_2, \ldots\ldots q_i], a) = \delta(q_1, a) \cup \delta(q_2, a) \cup \ldots\ldots \delta(q_i, a)$

$= [q_1, q_2, \ldots\ldots q_k]$ may be some state.

ii) Add the state $[q_1, q_2, \ldots\ldots q_k]$ to DFA if it is not already added in Q′.

iii) Then find the transitions for every input symbol from Σ for state $[q_1, q_2, \ldots\ldots q_k]$. If we get some state $[q_1, q_2, \ldots\ldots q_n]$ which is not in Q′ of DFA then add this state to Q′.

Scanned with OKEN Scanner

iv) If there is no new state generating then stop the process after finding all the transitions.

3) For the state $[q_1, q_2, \ldots\ldots q_n] \in Q'$ of DFA if any one state q_i is a final state of NFA then $[q_1, q_2, \ldots\ldots q_n]$ becomes a final state. Thus the set of all the final states $\in F'$ of DFA.

Problem based on NFA to DFA

Example 1.8.1 *Let $M = (\{q_0, q_1\}, \{0,1\}, \delta, q_0, \{q_1\})$ be NFA where $\delta(q_0, 0) = \{q_0, q_1\}, \delta(q_0, 1) = \{q_1\}$, $\delta(q_1, 0) = \phi, \delta(q_1, 1) = \{q_0, q_1\}$. Construct its equivalent DFA.* **SPPU : May-11, Marks 6**

Solution : Let the DFA $M' = (Q', \Sigma, \delta', q_0', F')$.

Now, the δ' function will be computed as follows -

As $\delta(q_0, 0) = \{q_0, q_1\}$ $\delta'([q_0], 0) = [q_0, q_1]$

As in NFA the initial state is q_0, the DFA will also contain the initial state $[q_0]$.

Let us draw the transition table for δ function for a given NFA.

	State \ Input	0	1	
$\delta(q_0, 0) \Rightarrow$	$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_1\}$	$\Leftarrow \delta(q_0, 1)$
$\delta(q_1, 0) \Rightarrow$	q_1	ϕ	$\{q_0, q_1\}$	$\Leftarrow \delta(q_1, 1)$

δ function for NFA

From the transition table we can compute that there are $[q_0], [q_1], [q_0, q_1]$ states for its equivalent DFA. We need to compute the transition from state $[q_0, q_1]$.

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

So, $$\delta'([q_0, q_1], 0) = [q_0, q_1]$$

Similarly,

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

So, $$\delta'([q_0, q_1], 1) = [q_0, q_1]$$

As in the given NFA q_1 is a final state, then in DFA wherever q_1 exists that state becomes a final state. Hence in the DFA final states are $[q_1]$ and $[q_0, q_1]$. Therefore set of final states $F = \{[q_1], [q_0, q_1]\}$.

The equivalent DFA is -

State \ Input	0	1
$\rightarrow q_0$	$[q_0, q_1]$	$[q_1]$
$[q_1]$	ϕ	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$

Transition diagram for equivalent DFA

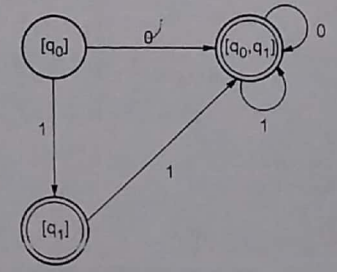


Fig. 1.8.1

Even we can change the names of the states of DFA.

$A = [q_0]$

$B = [q_1]$

$C = [q_0, q_1]$

With these new names the DFA will be as follows -

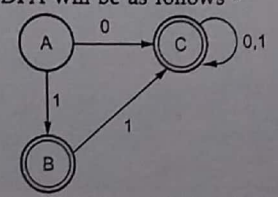


Fig. 1.8.2 An equivalent DFA

Scanned with OKEN Scanner

Example 1.8.2 *Construct DFA equivalent to the given NFA.* **SPPU : Dec.-10, Marks 6**

State \ Input	0	1
→ p	{p, q}	p
q	r	r
r	s	–
s	s	s

Solution : The NFA M = ({p, q, r, s}, {0, 1}, δ {p}, {s})

The equivalent DFA will be constructed.

State \ Input	0	1
→ [p]	[p, q]	[p]
[q]	[r]	[r]
[r]	[s]	–
[s]	[s]	[s]
[p, q]	[p, q, r]	[p, r]

Continuing with the generated new states.

State \ Input	0	1
→ [p]	[p, q]	[p]
[q]	[r]	[r]
[r]	[s]	–
[s]	[s]	[s]
[p, q]	[p, q, r]	[p, r]
[p, q, r]	[p, q, r, s]	[p, r]
[p, r]	[p, q, s]	[p]
[p, q, r, s]	[p, q, r, s]	[p, r, s]
[p, q, s]	[p, q, r, s]	[p, r, s]
[p, r, s]	[p, q, s]	[p, s]
[p, s]	[p, q, s]	[p, s]

The final state F' = { [s], [p, q, r, s], [p, q, s], [p, r, s], [p, s] }

The transition graph shows two disconnected parts. But part I will be accepted as final DFA because it consists of start state and final states, in part II there is no start state.

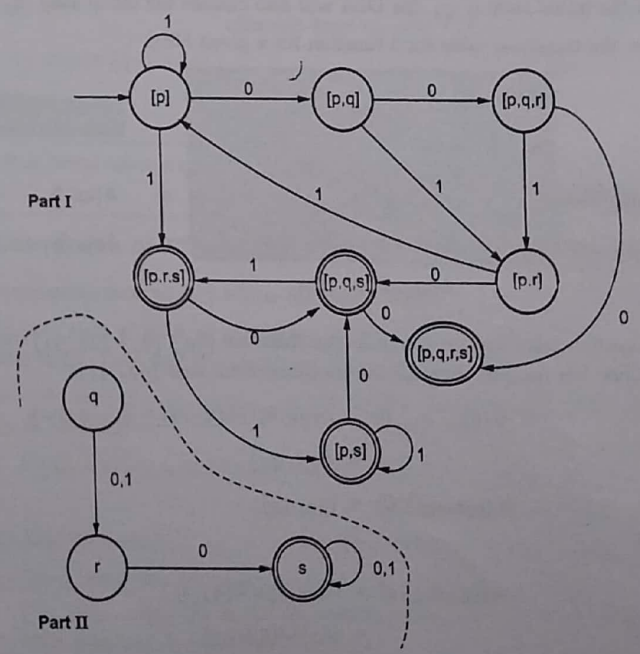


Fig. 1.8.3

Scanned with OKEN Scanner

Example 1.8.3 *Convert the following NFA into DFA.*

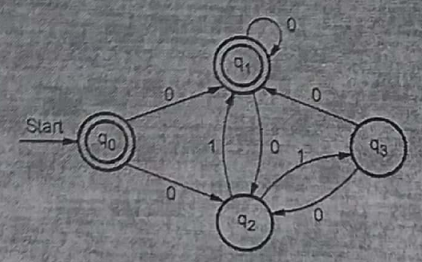


Fig. 1.8.4

Solution : We will first design a transition table from given transition diagram.

State \ Input	0	1
q_0	$\{q_1, q_2\}$	ϕ
q_1	$\{q_1, q_2\}$	ϕ
q_2	ϕ	$\{q_1, q_3\}$
q_3	$\{q_1, q_2\}$	ϕ

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

We can write this as

$$\delta'([q_0], 0) = [q_1, q_2]$$

Here $[q_1, q_2]$ becomes one newly generated state when 0 input is received in state $[q_0]$.

Similarly,

$\delta'([q_0], 1) = \phi$ No state getting generated.

$\delta(q_1, 0) = \boxed{\{q_1, q_2\}}$ we can write it as

$\delta'([q_1], 0) = [q_1, q_2]$

$\delta'([q_1], 1) = \phi$

$\delta'([q_2], 0) = \phi$

$\delta'(q_2, 1) = \{q_1, q_3\}$

i.e. $\delta'([q_2], 1) = \boxed{[q_1, q_3]}$ again a new state $[q_1, q_3]$ gets generated.

Similarly,

$$\delta'([q_3], 0) = [q_1, q_2]$$
$$\delta'([q_3], 1) = \phi$$

Now we will obtain δ' transitions on newly generated states $[q_1, q_2]$ and $[q_1, q_3]$.

$$\delta'([q_1, q_2], 0) = \delta([q_1], 0) \cup \delta([q_2], 0)$$
$$= \{q_1, q_2\} \cup \phi$$
$$= \{q_1, q_2\}$$
$$\therefore \quad \delta'([q_1, q_2], 0) = [q_1, q_2]$$

Similarly,

$$\delta'([q_1, q_2], 1) = \delta([q_1], 1) \cup \delta([q_2], 1)$$
$$= \phi \cup \{q_1, q_3\}$$
$$= \{q_1, q_3\}$$
$$\therefore \quad \delta'([q_1, q_2], 1) = [q_1, q_3]$$

Now,
$$\delta'([q_1, q_3], 0) = \delta([q_1], 0) \cup \delta([q_3], 0)$$
$$= \{q_1, q_2\} \cup \{q_1, q_2\}$$
$$= \{q_1, q_2\}$$
$$\therefore \quad \delta'([q_1, q_3], 0) = [q_1, q_2]$$

Similarly,

$$\delta'([q_1, q_3], 1) = \delta([q_1], 1) \cup \delta([q_3], 1)$$
$$= \phi \cup \phi$$
$$= \phi$$
$$\therefore \quad \delta'([q_1, q_3], 1) = \phi$$

As now no new states are getting generated.

Scanned with OKEN Scanner

The transition table for DFA using above δ′ transitions is

State \ Input	0	1
→ $[q_0]$	$[q_1, q_2]$	ϕ
$[q_1]$	$[q_1, q_2]$	ϕ
$[q_2]$	ϕ	$[q_1, q_3]$
$[q_3]$	$[q_1, q_2]$	ϕ
$[q_1, q_2]$	$[q_1, q_2]$	$[q_1, q_3]$
$[q_1, q_3]$	$[q_1, q_2]$	ϕ

The transition diagram can be -

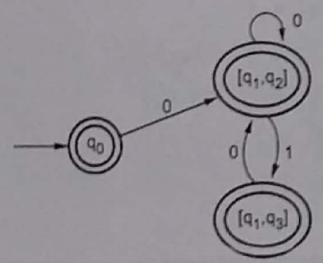
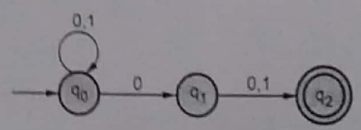


Fig. 1.8.5

As states q_0 and q_1 are final states original NFA states containing q_1 i.e. $[q_1, q_3]$ and $[q_1, q_2]$ are also final states.

Example 1.8.4 *Design an FA for the languages that contain strings with next-to-last symbol 0.*

Solution : We will construct NFA for the given problem



This NFA can be converted to DFA for that - first construct the transition table for above NFA.

State \ i/p	0	1
→ q_0	(q_0, q_1)	q_0
q_1	q_2	q_2
q_2	ϕ	ϕ

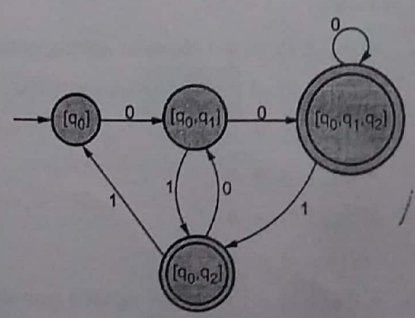
$\delta(q_0, 0) = (q_0, q_1)$ New state
$\delta(q_0, 1) = q_0$
$\delta(q_1, 0) = q_2$
$\delta(q_1, 1) = q_2$
$\delta(q_2, 0) = \phi$
$\delta(q_2, 1) = \phi$

Apply input transitions on new state

$\delta([q_0, q_1], 0) = (q_0, q_1, q_2)$ New state
$\delta([q_0, q_1], 1) = (q_0, q_2)$ New state
$\therefore \quad \delta([q_0, q_1, q_2], 0) = [q_0, q_1, q_2]$
$\delta([q_0, q_1, q_2], 1) = [q_0, q_2]$
$\delta([q_0, q_2], 0) = [q_0, q_1]$
$\delta([q_0, q_2]) = [q_0]$

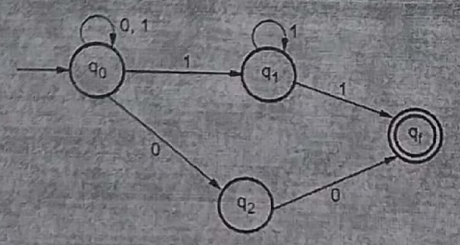
As no new states are getting generated, we stop applying input transitions. The DFA can be

State \ i/p	0	1
→ $[q_0]$	$[q_0, q_1]$	$[q_0]$
$[q_1]$	$[q_2]$	$[q_2]$
$[q_2]$	ϕ	ϕ
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0, q_2]$
* $[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$	$[q_0, q_2]$
* $[q_0, q_2]$	$[q_0, q_1]$	$[q_0]$



Scanned with OKEN Scanner

Example 1.8.5 *Obtain DFA equivalent to NFA.* **SPPU : May-16, End Sem., Marks 6**



Solution :

Step 1 : We will first draw the transition table for given transition diagram.

State \ Input	0	1
→ q_0	$\{q_0, q_2\}$	$\{q_0, q_1\}$
q_1	ϕ	$\{q_1, q_f\}$
q_2	$\{q_f\}$	ϕ
q_f	ϕ	ϕ

Step 2 : Now we will obtain δ transitions.

For each state on each input.

$\delta([q_0], 0) = \{q_0, q_2\} = [q_0, q_2]$ **New state**

$\delta([q_0], 1) = \{q_0, q_1\} = [q_0, q_1]$ **New state**

$\delta([q_1], 0) = \phi$

$\delta([q_1], 1) = \{q_1, q_f\} = [q_1, q_f]$ **New state**

$\delta([q_2], 0) = \{q_f\} = [q_f]$

$\delta([q_2], 1) = \phi$

$\delta([q_f], 0) = \delta([q_f], 1) = \phi$

Step 3 : Now we will obtain δ transitions on newly obtained states in step 2,

$\delta([q_0, q_2], 0) = \{q_0, q_2, q_f\} = [q_0, q_2, q_f]$ **New state**

$\delta([q_0, q_2], 1) = \{q_0, q_1\} = [q_0, q_1]$ **New state**

$\delta([q_0, q_1], 0) = \{q_0, q_2\} = [q_0, q_2]$

$\delta([q_0, q_1], 1) = \{q_0, q_1, q_f\} = [q_0, q_1, q_f]$ **New state**

$\delta([q_1, q_f], 0) = \phi$

$\delta([q_1, q_f], 1) = [q_1, q_f]$

Step 4 : Now we will obtain δ transitions on newly obtained states in step 3,

$\delta([q_0, q_2, q_f], 0) = [q_0, q_2, q_f]$

$\delta([q_0, q_2, q_f], 1) = [q_0, q_1]$

$\delta([q_0, q_1, q_f], 0) = [q_0, q_2]$

$\delta([q_0, q_1, q_f], 1) = [q_0, q_1, q_f]$

As now no new state is obtained, we will build the transition table as follows :

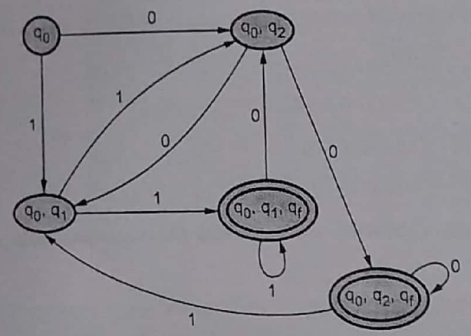
Step 5 :

State \ Input	0	1
$[q_0]$	$[q_0, q_2]$	$[q_0, q_1]$
$[q_1]$	ϕ	$[q_1, q_f]$
$[q_2]$	$[q_f]$	ϕ
$[q_f]$	ϕ	ϕ
$[q_0, q_2]$	$[q_0, q_2, q_f]$	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_2]$	$[q_0, q_1, q_f]$
$[q_1, q_f]$	ϕ	$[q_1, q_f]$
$[q_0, q_2, q_f]$	$[q_0, q_2, q_f]$	$[q_0, q_1]$
$[q_0, q_1, q_f]$	$[q_0, q_2]$	$[q_0, q_1, q_f]$

Clearly from above transition table, q_1, q_2, q_1, q_f and q_f are non-reachable states from start state. So we will eliminate them.

Scanned with OKEN Scanner

Step 6 : The DFA will then be as follows :



1.9 Conversion of NFA with Epsilon Moves to NFA without Epsilon

SPPU : May-12, Aug.-15, Oct.-16, Marks 8

In this method we try to remove all the ε transitions from given NFA. The method will be

1. Find out all the ε transitions from each state from Q. That will be called as ε - closure $\{q_i\}$ where $q_i \in Q$.
2. Then δ′ transitions can be obtained. The δ′ transitions means an ε-closure on δ moves.
3. Step - 2 is repeated for each input symbol and for each state of given NFA.
4. Using the resultant states the transition table for equivalent NFA without ε can be built.

Rule for conversion

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

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

Problems based on NFA with Epsilon to NFA without Epsilon

Example 1.9.1 *Convert the given NFA with ε to NFA without ε.* **SPPU : May-12, Marks 8**

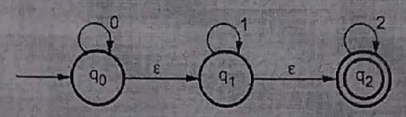


Fig. 1.9.1

Solution : We will first obtain ε - closure of each state i.e. we will find out ε - reachable states from current state.

Hence

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

As ε - closure (q_0) means with null input (no input symbol) we can reach to q_0, q_1 or q_2. In a similar manner for q_1 and q_2 ε - closures are obtained. Now we will obtain δ′ transitions for each state on each input symbol.

$$\delta'(q_0, 0) = \varepsilon \text{ - closure}\left(\delta\left(\hat{\delta}(q_0,\varepsilon), 0\right)\right)$$
$$= \varepsilon \text{ - closure}\left(\delta(\varepsilon \text{ - closure}(q_0), 0)\right)$$
$$= \varepsilon \text{ - closure}\left(\delta(q_0, q_1, q_2), 0\right)$$
$$= \varepsilon \text{ - closure}\left(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)\right)$$
$$= \varepsilon \text{ - closure}\left(q_0 \cup \phi \cup \phi\right)$$
$$= \varepsilon \text{ - closure}\left(q_0\right) = \{q_0, q_1, q_2\}$$
$$\delta'(q_0, 1) = \varepsilon \text{ - closure}\left(\delta\left(\hat{\delta}(q_0,\varepsilon), 1\right)\right)$$
$$= \varepsilon \text{ - closure}\left(\delta(q_0, q_1, q_2), 1\right)$$
$$= \varepsilon \text{ - closure}\left(\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)\right)$$
$$= \varepsilon \text{ - closure}\left(\phi \cup q_1 \cup \phi\right)$$
$$= \varepsilon \text{ - closure}\left(q_1\right)$$
$$\delta'(q_0, 1) = \{q_1, q_2\}$$
$$\delta'(q_1, 0) = \varepsilon \text{ - closure}\left(\delta\left(\hat{\delta}(q_1,\varepsilon), 0\right)\right)$$
$$= \varepsilon \text{ - closure}\left(\delta(\varepsilon \text{ - closure}(q_1), 0)\right)$$
$$= \varepsilon \text{ - closure}\left(\delta(q_1, q_2), 0\right)$$
$$= \varepsilon \text{ - closure}\left(\delta(q_1, 0) \cup \delta(q_2, 0)\right)$$
$$= \varepsilon \text{ - closure}\left(\phi \cup \phi\right)$$
$$= \varepsilon \text{ - closure}\left(\phi\right)$$
$$= \phi$$
$$\delta'(q_1, 1) = \varepsilon \text{ - closure}\left(\delta\left(\hat{\delta}(q_1,\varepsilon), 1\right)\right)$$

Scanned with OKEN Scanner

$$= \varepsilon\text{ - closure }(\delta(\varepsilon\text{ - closure}(q_1), 1))$$
$$= \varepsilon\text{ - closure }(\delta(q_1, q_2), 1)$$
$$= \varepsilon\text{ - closure }(\delta(q_1, 1) \cup \delta(q_2, 1))$$
$$= \varepsilon\text{ - closure }(q_1 \cup \phi)$$
$$= \varepsilon\text{ - closure }(q_1)$$
$$= \{q_1, q_2\}$$

$$\delta'(q_2, 0) = \varepsilon\text{ - closure }\left(\delta\left(\hat{\delta}(q_2, \varepsilon), 0\right)\right)$$
$$= \varepsilon\text{ - closure }(\delta(\varepsilon\text{ - closure }(q_2), 0))$$
$$= \varepsilon\text{ - closure }(\delta(q_2, 0))$$
$$= \varepsilon\text{ - closure }(\phi)$$
$$= \phi$$

$$\delta'(q_2, 1) = \varepsilon\text{ - closure }\left(\delta\left(\hat{\delta}(q_2, \varepsilon), 1\right)\right)$$
$$= \varepsilon\text{ - closure }(\delta(\varepsilon\text{ - closure}(q_2), 1))$$
$$= \varepsilon\text{ - closure }(\delta(q_2, 1))$$
$$= \varepsilon\text{ - closure }(\phi)$$
$$\delta'(q_2, 1) = \phi$$

$$\delta'(q_0, 2) = \varepsilon\text{ - closure }\left(\delta\left(\hat{\delta}(q_0, \varepsilon), 2\right)\right)$$
$$= \varepsilon\text{ - closure }(\delta(\varepsilon\text{ - closure}(q_0), 2))$$
$$= \varepsilon\text{ - closure }(\delta(q_0, q_1, q_2), 2)$$
$$= \varepsilon\text{ - closure }(\delta(q_0, 2) \cup \delta(q_1, 2) \cup \delta(q_2, 2))$$
$$= \varepsilon\text{ - closure }(\phi \cup \phi \cup q_2)$$
$$= \varepsilon\text{ - closure }(q_2)$$
$$= \{q_2\}$$

$$\delta'(q_1, 2) = \varepsilon\text{ - closure }\left(\delta\left(\hat{\delta}(q_1, \varepsilon), 2\right)\right)$$
$$= \varepsilon\text{ - closure }(\delta(\varepsilon\text{ - closure }(q_1), 2))$$
$$= \varepsilon\text{ - closure }(\delta(q_1, q_2), 2)$$
$$= \varepsilon\text{ - closure }(\delta(q_1, 2) \cup \delta(q_2, 2))$$
$$= \varepsilon\text{ - closure }(\phi \cup q_2)$$

$$= \{q_2\}$$

$$\delta'(q_2, 2) = \varepsilon\text{ - closure }\left(\delta\left(\hat{\delta}(q_2, \varepsilon), 2\right)\right)$$
$$= \varepsilon\text{ - closure }(\delta(\varepsilon\text{ - closure}(q_2), 2))$$
$$= \varepsilon\text{ - closure }(\delta(q_2, 2))$$
$$= \varepsilon\text{ - closure }(q_2)$$
$$= \{q_2\}$$

Now we will summarize all the computed δ' transitions -

$\delta'(q_0, 0) = \{q_0, q_1, q_2\}$, $\delta'(q_0, 1) = \{q_1, q_2\}$, $\delta'(q_0, 2) = \{q_2\}$

$\delta'(q_1, 0) = \phi$, $\delta'(q_1, 1) = \{q_1, q_2\}$, $\delta'(q_1, 2) = \{q_2\}$

$\delta'(q_2, 0) = \phi$, $\delta'(q_2, 1) = \phi$, $\delta'(q_2, 2) = \{q_2\}$

From this we can write the transition table as -

State \ Input	0	1	2
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 NFA will be -

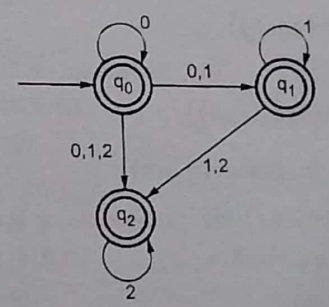


Fig. 1.9.2

Here q_0, q_1 and q_2 is a final state because ε - closure (q_0), ε - closure (q_1) and ε - closure (q_2) contains final state q_2.

Scanned with OKEN Scanner

Example 1.9.2 *Convert the following NFA with ε to NFA without ε.*

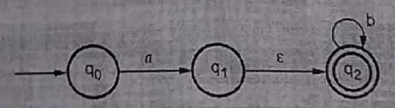


Fig. 1.9.3

Solution : We will first obtain ε - closures of q_0, q_1 and q_2 as follows.

ε - closure $(q_0) = \{q_0\}$

ε - closure $(q_1) = \{q_1, q_2\}$

ε - closure $(q_2) = \{q_2\}$

Now the δ' transitions on each input symbol is obtained as

$$\delta'(q_0, a) = \varepsilon\text{ - closure}\left(\delta\left(\hat{\delta}(q_0, \varepsilon), a\right)\right)$$
$$= \varepsilon\text{ - closure}(\delta(\varepsilon\text{ - closure}(q_0), a))$$
$$= \varepsilon\text{ - closure}(\delta(q_0, a))$$
$$= \varepsilon\text{ - closure}(q_1)$$
$$= \{q_1, q_2\}$$

$$\delta'(q_0, b) = \varepsilon\text{ - closure}\left(\delta\left(\hat{\delta}(q_0, \varepsilon), b\right)\right)$$
$$= \varepsilon\text{ - closure}(\delta(\varepsilon\text{ - closure}(q_0), b))$$
$$= \varepsilon\text{ - closure}(\delta(q_0, b))$$
$$= \phi$$

$$\delta'(q_1, a) = \varepsilon\text{ - closure}\left(\delta\left(\hat{\delta}(q_1, \varepsilon), a\right)\right)$$
$$= \varepsilon\text{ - closure}(\delta(\varepsilon\text{ - closure}(q_1), a))$$
$$= \varepsilon\text{ - closure}(\delta(q_1, q_2), a)$$
$$= \varepsilon\text{ - closure}(\delta(q_1, a) \cup \delta(q_2, a))$$
$$= \varepsilon\text{ - closure}(\phi \cup \phi)$$
$$= \phi$$

$$\delta'(q_1, b) = \varepsilon\text{ - closure}\left(\delta\left(\hat{\delta}(q_1, \varepsilon), b\right)\right)$$
$$= \varepsilon\text{ - closure}(\delta(\varepsilon\text{ - closure}(q_1), b))$$
$$= \varepsilon\text{ - closure}(\delta(q_1, q_2), b)$$

$$= \varepsilon\text{ - closure}(\delta(q_1, b) \cup \delta(q_2, b))$$
$$= \varepsilon\text{ - closure}(\phi \cup q_2)$$
$$= \varepsilon\text{ - closure}(q_2)$$
$$= \{q_2\}$$

$$\delta'(q_2, a) = \varepsilon\text{ - closure}\left(\delta\left(\hat{\delta}(q_2, \varepsilon), a\right)\right)$$
$$= \varepsilon\text{ - closure}(\delta(\varepsilon\text{ - closure}(q_2), a))$$
$$= \varepsilon\text{ - closure}(\delta(q_2, a))$$
$$= \varepsilon\text{ - closure}(\phi)$$
$$= \phi$$

$$\delta'(q_2, b) = \varepsilon\text{ - closure}\left(\delta\left(\hat{\delta}(q_2, \varepsilon), b\right)\right)$$
$$= \varepsilon\text{ - closure}(\delta(\varepsilon\text{ - closure}(q_2), b))$$
$$= \varepsilon\text{ - closure}(\delta(q_2, b))$$
$$= \varepsilon\text{ - closure}(q_2)$$
$$= \{q_2\}$$

The transition table can be -

State \ Input	a	b
→ q_0	$\{q_1, q_2\}$	ϕ
q_1	ϕ	$\{q_2\}$
q_2	ϕ	$\{q_2\}$

States q_1 and q_2 becomes the final as ε - closures of q_1 and q_2 contains the final state q_2. The NFA can be shown by following transition diagram -

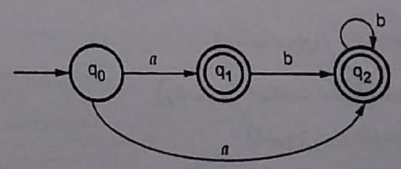
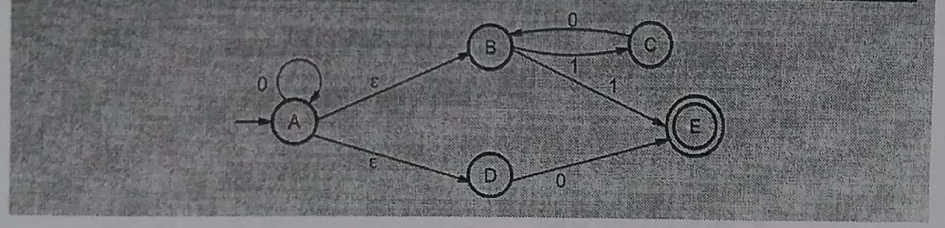


Fig. 1.9.4

Scanned with OKEN Scanner

Example 1.9.3 *Convert the given NFA-ε to an NFA.* **SPPU ; Aug.-15, Marks 6**



Solution :

We will obtain ε-closure of each state

ε - closure (A) = { A, B, D }

ε - closure (B) = { B }

ε - closure (C) = { C }

ε - closure (D) = { D }

ε - closure (E) = { E }

Now δ′ transitions for each state each input

$\delta'(A,0)$ = ε - closure $(\delta(\hat{\delta}(A,\varepsilon),0))$

= ε - closure $(\delta(\varepsilon\text{-closure}(A),0))$

= ε - closure $(\delta(A,B,D),0))$

= ε - closure $(\delta(A,0)\cup\delta(B,0)\cup\delta(D,0))$

= ε - closure $(A\cup\phi\cup E)$

= ε - closure $(A)\cup$ ε - closure (E)

= $\{A, B, D\}\cup\{E\}$

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

$\delta'(A,1)$ = ε - closure $(\delta(\hat{\delta}(A,\varepsilon),1))$

$\delta'(A,1)$ = ε - closure $(\delta(\varepsilon-\text{closure}(A),1))$

= ε - closure $(\delta(A,B,D),1))$

= ε - closure $(\delta(A,1)\cup\delta(B,1)\cup\delta(D,1))$

= ε - closure $(\phi\cup(C,E)\cup\phi)$

= ε - closure $(C\cup E)$

= ε - closure $(C)\cup$ ε - closure (E)

= { C, E }

$\delta'(B,0)$ = ε - closure $(\delta$ (ε - closure (B), 0))

= ε - closure $(\delta(B,0))$

= ε - closure (ϕ)

= ϕ

$\delta'(B,1)$ = ε - closure $(\delta$ (ε - closure (B), 1))

= ε - closure $(\delta(B,1))$

= ε - closure (C, E)

= ε - closure $(C)\cup$ ε - closure (E)

$\delta'(B,1)$ = { C, E }

$\delta'(C,0)$ = ε - closure $(\delta$ (ε - closure (C), 0))

= ε - closure $(\delta(C,0))$

= ε - closure (B)

$\delta'(C,0)$ = B

$\delta'(C,1)$ = ε - closure $(\delta$ (ε - closure (C), 1))

= ε - closure $(\delta(C,1))$

$\delta'(C,1)$ = ϕ

$\delta'(D,0)$ = ε - closure $(\delta$ (ε - closure (D), 0))

= ε - closure $(\delta(D,0))$

= ε - closure (E)

$\delta'(D,0)$ = E

$\delta'(D,1)$ = ε - closure $(\delta$ (ε - closure (D), 1))

= ε - closure $(\delta(D,1))$

= ε - closure (ϕ)

$\delta'(D,1)$ = ϕ

$\delta'(E,0)$ = ε - closure $(\delta$ (ε - closure (E), 0))

= ε - closure $(\delta(E,0))$

= ε - closure (ϕ)

$\delta'(E,0)$ = ϕ

$\delta'(E,1)$ = ε - closure $(\delta$ (ε - closure (E), 1))

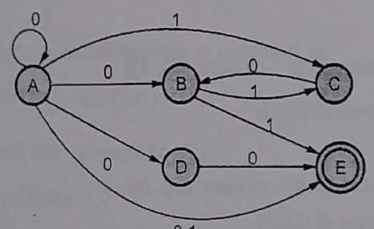
= ε - closure $(\delta(E,1))$

= ε - closure (ϕ)

$\delta'(E,1)$ = ϕ

Scanned with OKEN Scanner

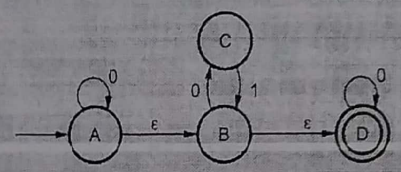
State \ Input	0	1
A	(A, B, D, E)	(C, E)
B	ϕ	(C, E)
C	B	ϕ
D	E	ϕ
E	ϕ	ϕ



NFA

Example 1.9.4 *Convert the following NFA-ε to equivalent NFA.*

SPPU : Oct.-16, In Sem, Marks 5



Solution :

Step 1 : We will first obtain ε - closure of each state.

ε - closure(A) = {A, B, D}

ε - closure(B) = {B, D}

ε - closure(D) = {D}

ε - closure(C) = {C}

Step 2 : Now obtain the δ′ transitions

$\delta'(A, 0)$ = ε - closure (δ ($\hat{\delta}$ (A, ε), 0))

= ε - closure (δ (ε - closure (A), 0))

= ε - closure (δ (A, B, D), 0))

= ε - closure (δ (A, 0) ∪ δ (B, 0) ∪ δ (D, 0))

= ε - closure (A ∪ C ∪ D)

= ε - closure (A) ∪ ε - closure (C) ∪ ε - closure (D)

= {A, B, D} ∪ {C} ∪ {D}

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

$\delta'(A, 1)$ = ε - closure (δ ($\hat{\delta}$ (A, ε), 1))

= ε - closure (δ (ε - closure (A), 1))

= ε - closure (δ (A, B, D), 1))

= ε - closure (δ (A, 1) ∪ δ (B, 1) ∪ δ (D, 1))

= ε - closure (ϕ ∪ ϕ ∪ ϕ)

$\delta'(A, 1)$ = ϕ

$\delta'(B, 0)$ = ε - closure (δ ($\hat{\delta}$ (B, ε), 0))

= ε - closure (δ (ε - closure (B), 0))

= ε - closure (δ (B, D), 0))

= ε - closure (δ (B, 0) ∪ δ (D, 0))

= ε - closure (C ∪ D)

= ε - closure (C) ∪ ε - closure (D)

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

$\delta'(B, 1)$ = ε - closure (δ ($\hat{\delta}$ (B, ε), 1))

= ε - closure (δ (ε - closure (B), 1))

= ε - closure (δ ((B, D), 1))

= ε - closure (δ (B, 1) ∪ δ (D, 1))

= ε - closure (ϕ ∪ ϕ)

$\delta'(B, 1)$ = ϕ

$\delta'(C, 0)$ = ε - closure (δ ($\hat{\delta}$ (C, ε), 0))

= ε - closure (δ (ε - closure (C), 0))

= ε - closure (δ (C, 0))

= ε - closure (ϕ)

Scanned with OKEN Scanner

$\delta'(C, 0) = \phi$

$\delta'(C, 1) = \varepsilon$ - closure $(\delta(\hat{\delta}(C, \varepsilon), 1))$

$= \varepsilon$ - closure $(\delta(\varepsilon$ - closure $(C), 1))$

$= \varepsilon$ - closure $(\delta(C, 1))$

$= \varepsilon$ - closure (B)

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

$\delta'(D, 0) = \varepsilon$ - closure $(\delta(\hat{\delta}(D, \varepsilon), 0))$

$= \varepsilon$ - closure $(\delta(\varepsilon$ - closure $(D), 0))$

$= \varepsilon$ - closure $(\delta(D, 0))$

$= \varepsilon$ - closure (D)

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

$\delta'(D, 1) = \varepsilon$ - closure $(\delta(\hat{\delta}(D, \varepsilon), 1))$

$= \varepsilon$ - closure $(\delta(\varepsilon$ - closure $(D), 1))$

$= \varepsilon$ - closure $(\delta(D, 1))$

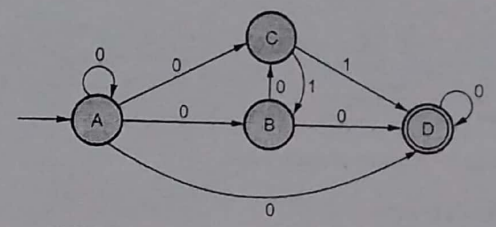
$= \varepsilon$ - closure (ϕ)

$\delta'(D, 1) = \{\phi\}$

Step 3 : From above computations, we can draw the transition table for all δ' transitions.

State \ Input	0	1
→ A	{A, B, C, D}	ϕ
B	{C, D}	ϕ
C	ϕ	{B, D}
(D)	{D}	ϕ

Step 4 : The transition diagram for NFA will be



1.10 Conversion of NFA with Epsilon Moves to DFA

SPPU : Oct.-16, Dec.-18, May-09,12, Marks 8

Method for converting NFA with ε to DFA

Step 1 : Consider $M = (Q, \Sigma, \delta, q_0, F)$ is a NFA with ε. We have to convert this NFA with ε to equivalent DFA denoted by

$$M_D = (Q_D, \Sigma, \delta_D, q_0, F_D)$$

Then obtain,

ε - closure$(q_0) = \{p_1, p_2, p_3, \ldots p_n\}$ then $[p_1, p_2, p_3, \ldots p_n]$ becomes a start state of DFA.

Now $[p_1, p_2, p_3, \ldots p_n] \in Q_D$

Step 2 : We will obtain δ transitions on $[p_1, p_2, p_3, \ldots p_n]$ for each input.

$$\delta_D([p_1, p_2, \ldots p_n], a) = \varepsilon \text{ - closure}(\delta(p_1, a) \cup \delta(p_2, a) \cup \ldots \delta(p_n, a))$$

$$= \bigcup_{i=1}^{n} \varepsilon \text{ - closure} \delta(p_i, a)$$

where a is input $\in \Sigma$.

Step 3 : The states obtained $[p_1, p_2, p_3, \ldots p_n] \in Q_D$. The states containing final state in p_i is a final state in DFA.

Problems based on NFA with Epsilon to DFA

Example 1.10.1 *Convert the given NFA into its equivalent DFA - or construct DFA that accepts the language represented by 0*1*2*, Make use of NFA.* **SPPU ; Oct.-16, In Sem.; Dec.-18, End Sem, Marks 8**

Scanned with OKEN Scanner

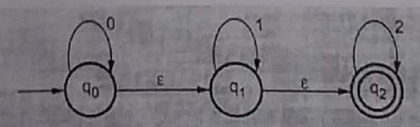


Fig. 1.10.1

Solution : Let us obtain ε - closure of each state.

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

ε - closure $(q_1) = \{q_1, q_2\}$

ε - closure $(q_2) = \{q_2\}$

Now we will obtain δ' transition. Let ε - closure $(q_0) = \{q_0, q_1, q_2\}$ call it as **state A.**

$$\delta'(A,0) = \varepsilon\text{ - closure } \{\delta((q_0,q_1,q_2),0)\}$$
$$= \varepsilon\text{ - closure } \{\delta(q_0,0) \cup \delta(q_1,0) \cup \delta(q_2,0)\}$$
$$= \varepsilon\text{ - closure } \{q_0\}$$
$$= \{q_0, q_1, q_2\} \quad \text{i.e. state A}$$

$$\delta'(A,1) = \varepsilon\text{ - closure } \{\delta((q_0,q_1,q_2),1)\}$$
$$= \varepsilon\text{ - closure } \{\delta(q_0,1) \cup \delta(q_1,1) \cup \delta(q_2,1)\}$$
$$= \varepsilon\text{ - closure } \{q_1\}$$
$$= \{q_1, q_2\} \quad \text{Call it as state B}$$

$$\delta'(A,2) = \varepsilon\text{ - closure } \{\delta((q_0,q_1,q_2),2)\}$$
$$= \varepsilon\text{ - closure } \{\delta(q_0,2) \cup \delta(q_1,2) \cup \delta(q_2,2)\}$$
$$= \varepsilon\text{ - closure } \{q_2\}$$
$$= \{q_2\} \quad \text{Call it as state C.}$$

Thus we have obtained

$\delta'(A,0) = A$

$\delta'(A,1) = B$

$\delta'(A,2) = C$

i.e.

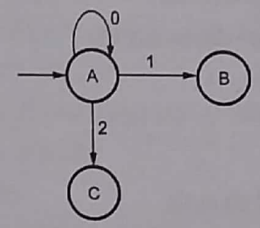


Fig. 1.10.2

Now we will find transitions on states B and C for each input.

Hence

$$\delta'(B,0) = \varepsilon\text{ - closure } \{\delta(q_1,q_2),0\}$$
$$= \varepsilon\text{ - closure } \{\delta(q_1,0) \cup \delta(q_2,0)\}$$
$$= \varepsilon\text{ - closure } \{\phi\}$$
$$= \phi$$

$$\delta'(B,1) = \varepsilon\text{ - closure } \{\delta(q_1,q_2),1\}$$
$$= \varepsilon\text{ - closure } \{\delta(q_1,1) \cup \delta(q_2,1)\}$$
$$= \varepsilon\text{ - closure } \{q_1\}$$
$$= \{q_1, q_2\} \text{ i.e state B itself.}$$

$$\delta'(B,2) = \varepsilon\text{ - closure } \{\delta(q_1,q_2),2\}$$
$$= \varepsilon\text{ - closure } \{\delta(q_1,2) \cup \delta(q_2,2)\}$$
$$= \varepsilon\text{ - closure } \{q_2\}$$
$$= \{q_2\} \text{ i.e state C.}$$

Hence

$\delta'(B,0) = \phi$

$\delta'(B,1) = B$

$\delta'(B,2) = C$

Scanned with OKEN Scanner

The partial transition diagram will be

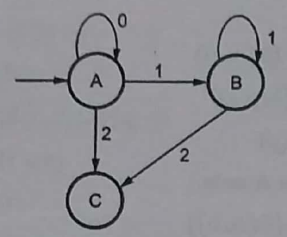


Fig. 1.10.3

Now we will obtain transitions for C :

$$\delta'(C,0) = \varepsilon\text{ - closure }\{\delta(q_2,0)\}$$
$$= \varepsilon\text{ - closure }\{\phi\}$$
$$= \phi$$
$$\delta'(C,1) = \varepsilon\text{ - closure }\{\delta(q_2,1)\}$$
$$= \varepsilon\text{ - closure }\{\phi\}$$
$$= \phi$$
$$\delta'(C,2) = \varepsilon\text{ - closure }\{\delta(q_2,2)\}$$
$$= q_2$$

Hence the DFA is

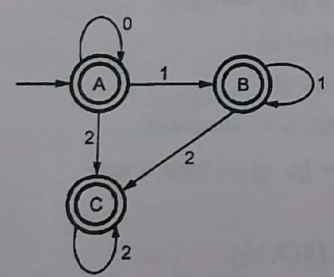


Fig. 1.10.4

As $A = \{q_0, q_1, q_2\}$ in which final state q_2 lies hence A is final state in $B = \{q_1, q_2\}$ the state q_2 lies hence B is also final state in $C = \{q_2\}$, the state q_2 lies hence C is also a final state.

Example 1.10.2 *Convert the given NFA with ε to its equivalent DFA.*

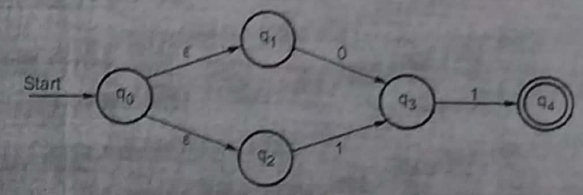


Fig. 1.10.5

Solution :

$$\varepsilon\text{ - closure }\{q_0\} = \{q_0, q_1, q_2\}$$
$$\varepsilon\text{ - closure }\{q_1\} = \{q_1\}$$
$$\varepsilon\text{ - closure }\{q_2\} = \{q_2\}$$
$$\varepsilon\text{ - closure }\{q_3\} = \{q_3\}$$
$$\varepsilon\text{ - closure }\{q_4\} = \{q_4\}$$

Now, Let ε - closure $\{q_0\} = \{q_0, q_1, q_2\}$ be state A.

Hence

$$\delta'(A,0) = \varepsilon\text{ - closure }\{\delta((q_0, q_1, q_2),0)\}$$
$$= \varepsilon\text{ - closure }\{\delta(q_0,0) \cup \delta(q_1,0) \cup \delta(q_2,0)\}$$
$$= \varepsilon\text{ - closure }\{q_3\}$$
$$= \{q_3\} \text{ call it as state B.}$$
$$\delta'(A,1) = \varepsilon\text{ - closure }\{\delta((q_0, q_1, q_2),1)\}$$
$$= \varepsilon\text{ - closure }\{\delta(q_0,1) \cup \delta(q_1,1) \cup \delta(q_2,1)\}$$
$$= \varepsilon\text{ - closure }\{q_3\} = q_3 = B.$$

The partial DFA will be

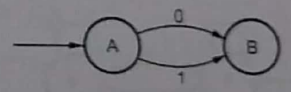


Fig. 1.10.6

Now,

$$\delta'(B,0) = \varepsilon\text{ - closure }\{\delta(q_3,0)\}$$
$$= \phi$$
$$\delta'(B,1) = \varepsilon\text{ - closure }\{\delta(q_3,1)\}$$
$$= \varepsilon\text{ - closure }\{q_4\}$$
$$= \{q_4\} \text{ i.e. state C}$$
$$\delta'(C,0) = \varepsilon\text{ - closure }\{\delta(q_4,0)\}$$

Scanned with OKEN Scanner

$= \phi$

$\delta'(C,1) = \varepsilon$ - closure $\{\delta(q_4, 1)\}$

$= \phi$

The DFA will be,

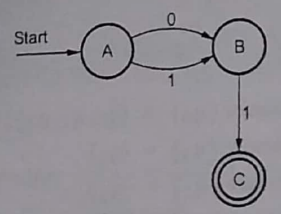


Fig. 1.10.7

Example 1.10.3 *Consider following NFA with ε.*

State \ Input	a	b	c	ε
→ p	{p}	{q}	{r}	ϕ
q	{q}	{r}	ϕ	{p}
r	{r}	ϕ	{p}	{q}

Convert it to its equivalent DFA. **SPPU : May-12, Marks 8**

Solution : We will first compute ε - closure for start state p.

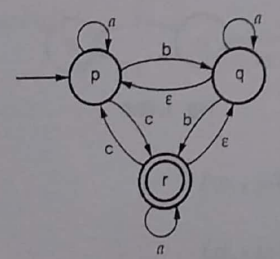


Fig. 1.10.8

ε - closure (p) = {p} call it as state A.

Now we will obtain δ transitions on state A.

State A

$\delta(A, a) = \varepsilon$ - closure $(\delta(A, a))$

$= \varepsilon$ - closure$(\delta(p, a))$

$= \varepsilon$ - closure (p)

$= \{p\}$ i.e. state A only.

$\delta(A, b) = \varepsilon$ - closure $(\delta(A, b))$

$= \varepsilon$ - closure$(\delta(p, b))$

$= \varepsilon$ - closure (q)

$= \{q, p\}$ i.e. $\{p, q\}$ Let us call it state B.

$\delta(A, b) = B$

$\delta(A, c) = \varepsilon$ - closure $(\delta(A, c))$

$= \varepsilon$ - closure $(\delta(p, c))$

$= \varepsilon$ - closure $(r) = \{q, r\}$. Call it as state C.

State B = {p, q}

$\delta(B, a) = \varepsilon$ - closure $(\delta(B, a))$

$= \varepsilon$ - closure$(\delta(p, q), a)$

$= \varepsilon$ - closure $(\delta(p, a) \cup \delta(q, a))$

$= \varepsilon$ - closure $(p \cup q)$

$= \varepsilon$ - closure (p, q)

$= \varepsilon$ - closure$(p) \cup \varepsilon$ - closure(q)

$= \{p\} \cup \{q\} = \{p, q\}$ i.e. state B only.

$\therefore \quad \delta(B, a) = B.$

$\delta(B, b) = \varepsilon$ - closure $(\delta(B, b))$

$= \varepsilon$ - closure$(\delta(p, q), b)$

$= \varepsilon$ - closure $(\delta(p, b) \cup \delta(q, b))$

$= \varepsilon$ - closure $(q \cup r)$

$= \varepsilon$ - closure (q, r)

$= \varepsilon$ - closure$(q) \cup \varepsilon$ - closure(r)

$= \{p, q\} \cup \{q, r\} = \{p, q, r\}$ i.e. state D.

Scanned with OKEN Scanner

$\therefore \quad \delta(B, b) = D.$

$\delta(B, c) = \varepsilon\text{ - closure } (\delta(B, c))$

$= \varepsilon\text{ - closure}(\delta(p, q), c)$

$= \varepsilon\text{ - closure } (\delta(p, c) \cup \delta(q, c))$

$= \varepsilon\text{ - closure } (r \cup \phi)$

$= \varepsilon\text{ - closure } (r)$

$= \{q, r\}$

$\delta(B, c) = C$

State C = {q, r}

$\delta(C, a) = \varepsilon\text{ - closure } (\delta(C, a))$

$= \varepsilon\text{ - closure}(\delta(q, r), a)$

$= \varepsilon\text{ - closure } (\delta(q, a) \cup \delta(r, a))$

$= \varepsilon\text{ - closure } (q \cup r)$

$= \varepsilon\text{ - closure } (q) \cup \varepsilon\text{ - closure}(r)$

$= \{p, q\} \cup \{q, r\}$

$= \{p, q, r\}$ i.e. state D.

$\therefore \quad \delta(C, a) = D$

$\delta(C, b) = \varepsilon\text{ - closure } (\delta(C, b))$

$= \varepsilon\text{ - closure}(\delta(q, r), b)$

$= \varepsilon\text{ - closure } (\delta(q, b) \cup \delta(r, b))$

$= \varepsilon\text{ - closure } (r \cup \phi)$

$= \varepsilon\text{ - closure } (r)$

$= \{q, r\}$ i.e. state C.

$\therefore \quad \delta(C, b) = C$

$\delta(C, c) = \varepsilon\text{ - closure } (\delta(C, c))$

$= \varepsilon\text{ - closure}(\delta(q, r), c)$

$= \varepsilon\text{ - closure } (\delta(q, c) \cup \delta(r, c))$

$= \varepsilon\text{ - closure } (\phi \cup p)$

$= \varepsilon\text{ - closure } (p)$

$= \{p\}$ i.e. state A.

$\therefore \quad \delta(C, c) = A$

State D = {p, q, r}

$\delta(D, a) = \varepsilon\text{ - closure } (\delta(D, a))$

$= \varepsilon\text{ - closure}(\delta(p, q, r), a)$

$= \varepsilon\text{ - closure } (\delta(p, a) \cup \delta(q, a) \cup \delta(r, a))$

$= \varepsilon\text{ - closure } (p \cup q \cup r)$

$= \varepsilon\text{ - closure } (p) \cup \varepsilon\text{ - closure}(q) \cup \varepsilon\text{ - closure}(r)$

$= \{p, q, r\}$ i.e. state D.

$\therefore \quad \delta(D, a) = D$

$\delta(D, b) = \varepsilon\text{ - closure } (\delta(D, b))$

$= \varepsilon\text{ - closure}(\delta(p, q, r), b)$

$= \varepsilon\text{ - closure } (\delta(p, b) \cup \delta(q, b) \cup \delta(r, b))$

$= \varepsilon\text{ - closure } (q \cup r \cup \phi)$

$= \varepsilon\text{ - closure}(q, r)$

$= \varepsilon\text{ - closure } (q) \cup \varepsilon\text{ - closure}(r)$

$= \{p, q, r\}$ i.e. state D.

$\therefore \quad \delta(D, b) = D$

$\delta(D, c) = \varepsilon\text{ - closure } (\delta(D, c))$

$= \varepsilon\text{ - closure}(\delta(p, q, r), c)$

$= \varepsilon\text{ - closure } (\delta(p, c) \cup \delta(q, c) \cup \delta(r, c))$

$= \varepsilon\text{ - closure } (r \cup \phi \cup p)$

$= \varepsilon\text{ - closure } (r) \cup \varepsilon\text{ - closure}(p)$

$= \{q, r\} \cup \{p\}$

$= \{p, q, r\}$ i.e. state D.

$\therefore \quad \delta(D, c) = D$

Scanned with OKEN Scanner

The transition table from above calculations can be obtained as

State \ Input	a	b	c
→ A	A	B	C
B	B	D	C
C	D	C	A
D	D	D	D

As state A = {p} it is a start state and states C and D contain final state r, hence these are final states.

The transition diagram for the DFA is

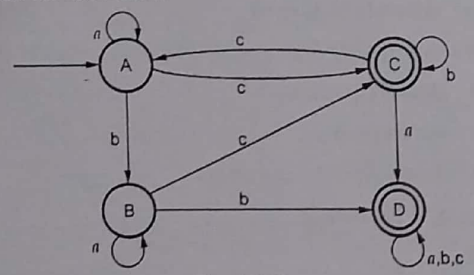
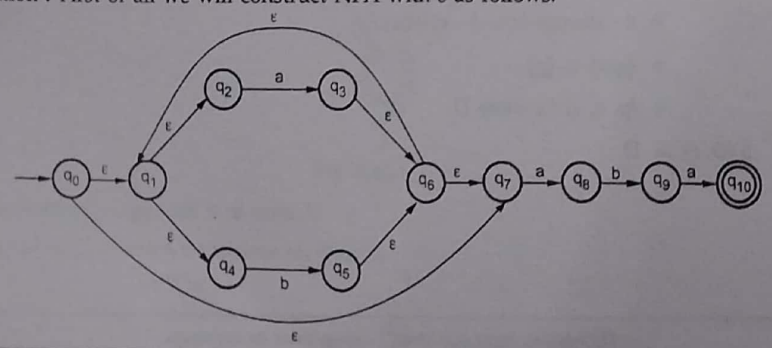


Fig. 1.10.9

Example 1.10.4 *Construct a NFA that accepts the set of all strings over {a, b} ending in aba. Use this NFA to construct DFA accepting the same set of strings.*

SPPU : May-09, Marks 8

Solution : First of all we will construct NFA with ε as follows.



Now we will convert this NFA with ε to DFA.

Let Start state

ε - closure $(q_0) = \{q_0, q_1, q_2, q_4, q_7\}$ → **Call it as state A**

Now we will find a and b transitions on state A.

$$\delta^1(A, a) = \varepsilon\text{ - closure }(\delta(A, a))$$

$$= \varepsilon\text{ - closure }(\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a) \cup \delta(q_4, a) \cup \delta(q_7\ a))$$

$$= \varepsilon\text{ - closure }(\phi \cup \phi \cup q_3 \cup \phi \cup q_8)$$

$$= \varepsilon\text{ - closure }(\{q_3, q_8\})$$

$$= (q_3, q_6, q_7, q_1, q_2, q_4, q_8) \text{ sorting it}$$

$$= (q_1, q_2, q_3, q_4, q_6, q_7, q_8) \rightarrow \textbf{Call it as B}$$

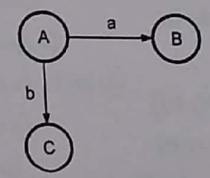
$$\delta'(A, b) = \varepsilon\text{ - closure }(\delta(A, b))$$

$$= \varepsilon\text{ - closure }(q_5)$$

$$= (q_5, q_6, q_1, q_2, q_4, q_7)$$

$$= (q_1, q_2, q_4, q_5, q_6, q_7) \rightarrow \textbf{Call it as C}$$

The partial DFA can be



Now consider state B for input a and b transitions.

$$\delta'(B, a) = \varepsilon\text{ - closure }(\delta(B, a))$$

$$= \varepsilon\text{ - closure }(\delta(q_1, a) \cup \delta(q_2, a) \cup \delta(q_3, a) \cup \delta(q_4, a) \cup \delta(q_6, a) \cup \delta(q_7, a) \cup \delta(q_8, a)$$

$$= \varepsilon\text{ - closure }(q_3, q_8)$$

$$= \{1, 2, 3, 4, 6, 7, 8\}$$

i.e. B

$$\delta'(B, b) = \varepsilon\text{ - closure }(\delta(B, b))$$

$$= \varepsilon\text{ - closure }(\delta(q_1, b) \cup \delta(q_2, b) \cup \delta(q_3, b) \cup \delta(q_4, b) \cup \delta(q_6, b) \cup \delta(q_7, b) \cup \delta(q_8, b))$$

Scanned with OKEN Scanner

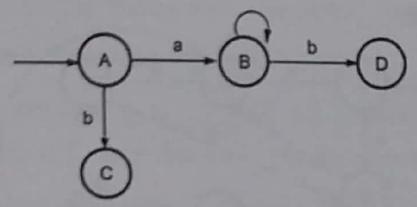
$= \varepsilon$ - closure (q_5, q_9)

$= (q_5, q_6, q_7, q_1, q_2, q_4, q_9)$ Sorting these states.

$= (q_1, q_2, q_4, q_5, q_6, q_7, q_9) \rightarrow$ **Call it as D state**

$= D$

The DFA partially can be drawn as



Now consider state C for input a and b transitions

$\delta'(C, a) = \varepsilon$ - closure $(\delta(C, a))$

$= \varepsilon$ - closure $(\delta(q_1, q_2, q_4, q_5, q_6, q_7), a)$

$= \varepsilon$ - closure $(\delta(q_1, a) \cup \delta(q_2, a) \cup \delta(q_4, a) \cup \delta(q_5, a) \cup \delta(q_6, a) \cup \delta(q_7, a))$

$= \varepsilon$ - closure $(\phi \cup q_3 \cup \phi \cup \phi \cup \phi \cup q_8)$

$= \varepsilon$ - closure (q_3, q_8)

$= (q_1, q_2, q_3, q_4, q_6, q_7, q_8)$

i.e. B state

$\delta'(C, b) = \varepsilon$ - closure $(\delta(c, b))$

$= \varepsilon$ - closure $(\delta((q_1, q_2, q_4, q_5, q_6, q_7), b))$

$= \varepsilon$ - closure $(\delta(q_1, b) \cup \delta(q_2, b) \cup \delta(q_4, b) \cup \delta(q_5, b) \cup \delta(q_6, b) \cup \delta(q_7, b))$

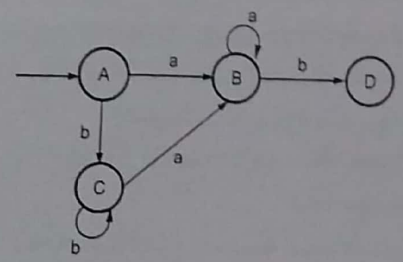
$= \varepsilon$ - closure $(\phi \cup \phi \cup q_5 \cup \phi \cup \phi \cup \phi)$

$= \varepsilon$ - closure (q_5)

$= (q_1, q_2, q_4, q_5, q_6, q_7)$

i.e. C state

Partially DFA can be



Now consider state D for transition on input a and b

$\delta'(D, a) = \varepsilon$ - closure $(\delta((q_1, q_2, q_4, q_5, q_6, q_7, q_9), a))$

$= \varepsilon$ - closure $(\delta(q_1, a) \cup \delta(q_2, a) \cup \delta(q_4, a) \cup \delta(q_5, a) \cup \delta(q_6, a) \cup \delta(q_7, a) \cup \delta(q_9, a))$

$= \varepsilon$ - closure $(\phi \cup q_3 \cup \phi \cup \phi \cup \phi \cup q_8 \cup q_{10})$

$= \varepsilon$ - closure (q_3, q_8, q_{10})

$= (q_3, q_6, q_1, q_2, q_4, q_8, q_{10})$ Sorting it

$= (q_1, q_2, q_3, q_4, q_6, q_8, q_{10}) \rightarrow$ Call it as E.

$\delta'(D, b) = \varepsilon$ - closure $(\delta((q_1, q_2, q_4, q_5, q_6, q_7, q_9), b))$

$= \varepsilon$ - closure $(\delta(q_1, b) \cup \delta(q_2, b) \cup \delta(q_4, b) \cup \delta(q_5, b) \cup \delta(q_6, b) \cup \delta(q_7, b) \cup \delta(q_9, b))$

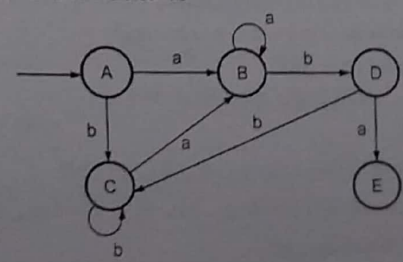
$= \varepsilon$ - closure $(\phi \cup \phi \cup q_5 \cup \phi \cup \phi \cup \phi \cup \phi)$

$= \varepsilon$ - closure (q_5)

$= (q_1, q_2, q_4, q_5, q_6, q_7)$

i.e. state C only

Partially the DFA can be constructed as



Scanned with OKEN Scanner

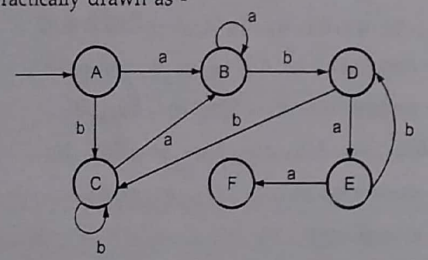
Now consider state E for transitions of input a and b.

$$\delta'(E, a) = \varepsilon\text{ - closure } (\delta((q_1, q_2, q_3, q_4, q_6, q_8, q_{10}), a))$$
$$= \varepsilon\text{ - closure } (\delta(q_1, a) \cup \delta(q_2, a) \cup \delta(q_3, a) \cup \delta(q_4, a) \cup \delta(q_6, a) \cup \delta(q_8, a) \cup \delta(q_{10}, a))$$
$$= \varepsilon\text{ - closure } (\phi \cup q_3 \cup \phi \cup \phi \cup \phi \cup \phi \cup \phi)$$
$$= \varepsilon\text{ - closure } (q_3)$$
$$= (q_1, q_2, q_3, q_4, q_6, q_7) \rightarrow \text{Call it as state F.}$$

$$\delta'(E, b) = \varepsilon\text{ - closure } (\delta((q_1, q_2, q_3, q_4, q_6, q_8, q_{10}), b))$$
$$= \varepsilon\text{ - closure } (\delta(q_1, b) \cup \delta(q_2, b) \cup \delta(q_3, b) \cup \delta(q_4, b) \cup \delta(q_6, b) \cup \delta(q_8, b) \cup \delta(q_{10}, b))$$
$$= \varepsilon\text{ - closure } (\phi \cup \phi \cup \phi \cup q_5 \cup \phi \cup q_9 \cup \phi)$$
$$= \varepsilon\text{ - closure } (q_5, q_9)$$
$$= (q_1, q_2, q_4, q_5, q_6, q_7, q_9)$$

i.e. state D.

The DFA can be practically drawn as -



Now process state F.

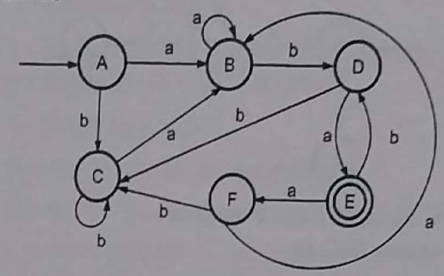
$$\delta'(F, a) = \varepsilon\text{ - closure } (\delta((q_1, q_2, q_3, q_4, q_6, q_7), a))$$
$$= \varepsilon\text{ - closure } (\delta(q_1, a) \cup \delta(q_2, a) \cup \delta(q_3, a) \cup \delta(q_4, a) \cup \delta(q_6, a) \cup \delta(q_7, a))$$
$$= \varepsilon\text{ - closure } (\phi \cup q_3 \cup \phi \cup \phi \cup \phi \cup q_8)$$
$$= \varepsilon\text{ - closure } (q_3, q_8)$$
$$= (q_1, q_2, q_3, q_4, q_6, q_7, q_8)$$

i.e. state B

$$\delta'(F, b) = \varepsilon\text{ - closure } (\delta((q_1, q_2, q_3, q_4, q_6, q_7), b))$$
$$= \varepsilon\text{ - closure } (\delta(q_1, b) \cup \delta(q_2, b) \cup (q_3, b) \cup \delta(q_4, b) \cup \delta(q_6, b) \cup \delta(q_7, b))$$

$$= \varepsilon\text{ - closure } (\phi \cup \phi \cup \phi \cup q_5 \cup \phi \cup \phi)$$
$$= \varepsilon\text{ - closure } (q_5)$$
$$= (q_1, q_2, q_4, q_5, q_6, q_7)$$
$$= \text{State C}$$

Finally the DFA can be



As state E contains final state q_{10}, in the resultant DFA, E is a final state.

1.11 Difference between NFA and DFA

SPPU : May-09,10, Aug.-17, Marks 2

Sr. No.	NFA	DFA
1.	NFA stands for Non deterministic Finite Automata.	DFA stands for Deterministic Finite Automata
2.	For every input symbol there can be more than one state transition.	For every input symbol there can be only one state transition.
3.	NFA can use empty string transition.	DFA can not use empty string transition.
4.	Construction of NFA is simple	Construction of DFA is difficult

University Question

1. *What is the difference between NFA and DFA ?*

SPPU : May-09, 10, Aug.-17(In Sem), Marks 2

1.12 Minimization of FA

SPPU : Dec.-12, Marks 8

The minimization of FSM means reducing the number of states from given FA. Thus we get the FSM with redundant states after minimizing the FSM.

Scanned with OKEN Scanner

While minimizing FSM we first find out which two states are equivalent we can represent those two states by one representative state.

Definition of equivalent states -

The two states q_1 and q_2 are equivalent if both $\delta(q_1, x)$ and $\delta(q_2, x)$ are final states or both of them are non final states for all $x \in \Sigma^*$ (Σ^* indicate any string of any length) we can minimize the given FSM by finding equivalent states.

Method for Construction of Minimum State Automata :

Step 1 : We will create a set π_0 as $\pi_0 = \{ Q_1^0, Q_2^0 \}$ where Q_1^0 is set of all final states and $Q_2^0 = Q - Q_1^0$ where Q is a set of all the states in DFA.

Step 2 : Now we will construct π_{K+1} from π_K. Let Q_i^K be any subset in π_K. If q_1 and q_2 are in Q_i^K they are (K + 1) equivalent provided $\delta(q_1, a)$ and $\delta(q_2, a)$ are K equivalent. Find out whether $\delta(q_1, a)$ and $\delta(q_2, a)$ are residing in the same equivalence class π_K. Then it is said that q_1 and q_2 are (K + 1) equivalent. Thus from Q_i^K we create (K + 1) equivalence classes. Repeat step 2 for every Q_i^K in π_K and obtain all the elements of π_{K+1}.

Step 3 : Construct π_n for n = 1, 2, ... until $\pi_n = \pi_{n+1}$.

Step 4 : Then replace all the equivalent states in one equivalence class by representative state. This helps in minimizing the given DFA.

Let us understand this method with the help of some examples.

Example 1.12.1 *Construct the minimum state automaton for the following transition diagram.*

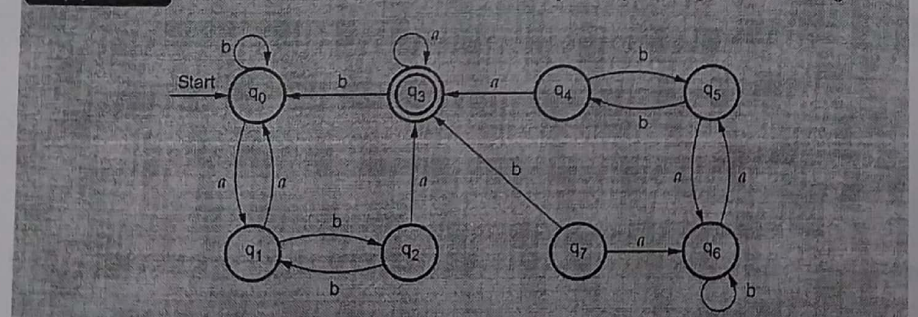


Fig. 1.12.1 **SPPU : Dec.-12, Marks 8**

Solution : We will first construct a transition table for the given DFA.

We will start constructing equivalence classes.

State \ Input	a	b
→ q_0	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_1
q_3	q_3	q_0
q_4	q_3	q_5
q_6	q_6	q_4
q_6	q_5	q_6
q_7	q_6	q_3

Step 1 : We will first find 0 - equivalence. For that purpose we will create two sets - one set containing all the final states and other set containing all the non-final states.

$\therefore$ 0 - Equivalent = $\{q_0, q_1, q_2, q_4, q_5, q_6, q_7\}, \{q_3\}$

Now will check equivalence among all the states of $\{q_0, q_1, q_2, q_4, q_5, q_6, q_7\}$ by means of δ mapping functions

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

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

Belong to same set $\quad$ Belong to same set

$\therefore$ q_0 and q_1 are equivalent. In this way we will consider every pair from the set $\{q_0, q_1, q_2, q_4, q_5, q_6, q_7\}$. Now consider pair $\{q_0, q_2\}$

$\delta(q_0, a) = q_1$

$\delta(q_2, a) = q_3$

Belong to different sets

$\therefore q_0$ and q_2 are not equivalent

Continuing in this way we get -

1 - equivalence = $\{q_0, q_1, q_5, q_6\}\{q_2, q_4\}, \{q_3\}, \{q_7\}$

Scanned with OKEN Scanner

Step 2 : Now will find 2 - equivalence from the sets obtained in 1 - equivalence.

Consider the set $\{q_0, q_1, q_5, q_6\}$ first. We will compare q_0 with q_1, q_5, q_6. Then we find

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

Belong to same set Belong to different set

$\therefore$ q_0 and q_1 now does not belong to same set.

Now compare states q_0, q_5 states.

$$\delta(q_0, a) = q_1 \qquad \delta(q_0, b) = q_0$$
$$\delta(q_5, a) = q_0 \qquad \delta(q_5, b) = q_4$$

Belong to different sets

$\therefore$ q_0, q_5 are not equivalent. Now we will compare q_1 and q_5.

$$\delta(q_1, a) = q_0 \qquad \delta(q_1, b) = q_2$$
$$\delta(q_5, a) = q_6 \qquad \delta(q_5, b) = q_4$$

Belong to same set Belong to same set

That means - q_1 and q_5 are equivalent. Hence the 2 - equivalence set can be written as -

$$\boxed{2\text{ - equivalence} = \{q_0, q_6\}, \{q_1, q_5\}, \{q_2, q_4\}, \{q_3\}, \{q_7\}}$$

Step 3 : We will compare $\{q_0, q_6\}$

$$\delta(q_0, a) = q_1 \qquad \delta(q_0, b) = q_0$$
$$\delta(q_6, a) = q_5 \qquad \delta(q_6, b) = q_6$$

Belong to same set Belong to same set

$\therefore$ $\{q_0, q_6\}$ are equivalent. Similarly, we will compare $\{q_1, q_5\}, \{q_2, q_4\}$ and we find them as equivalent. Hence 3 - equivalence set is as follows :

$$\boxed{3\text{ - equivalence} = \{q_0, q_6\}, \{q_1, q_5\}, \{q_2, q_4\}, \{q_3\}, \{q_7\}}$$

As 2 - equivalence and 3 - equivalence are same sets. We will stop finding further equivalences. Hence we can have minimized transition table and minimized DFA as follows.

State \ Input	a	b
→ $[q_0, q_6]$	$[q_1, q_5]$	$[q_0, q_6]$
$[q_1, q_5]$	$[q_0, q_6]$	$[q_2, q_4]$
$[q_2, q_4]$	$[q_3]$	$[q_1, q_5]$
* $[q_3]$	$[q_3]$	$[q_0, q_6]$
$[q_7]$	$[q_0, q_6]$	$[q_3]$

The transition diagram with minimized states is -

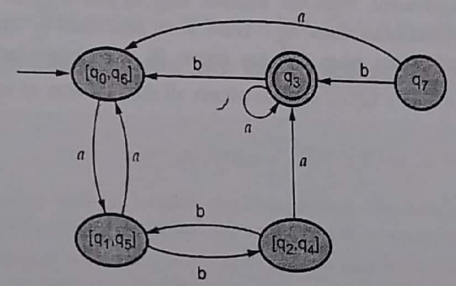


Fig. 1.12.2

1.13 Equivalence of FAs

- The two finite automata are said to be equivalent if both the automata accept the same set of strings, over an input set Σ.
- When two FAs are equivalent then there is some string x over Σ, on acceptance of that string if one FA reaches to final state other FA also reaches to final state.
- We can compare whether two FAs are equivalent or not using following method.

Method for Comparing two FAs

Let M and M' be two FAs and Σ is a set of input strings.

1. We will construct a transition table have pairwise entries (q, q') where $q \in M$ and $q' \in M'$ for each input symbol.
2. If we get in a pair as one final state and other nonfinal state then we terminate construction of transition table declaring that two FAs are not equivalent.

Scanned with OKEN Scanner

3. The construction of transition table gets terminated when there is no new pair appearing in the transition table.

Let us take one example to understand the technique of comparing two FAs.

Example 1.13.1 *Consider the DFAs given below. Are they equivalent ?*

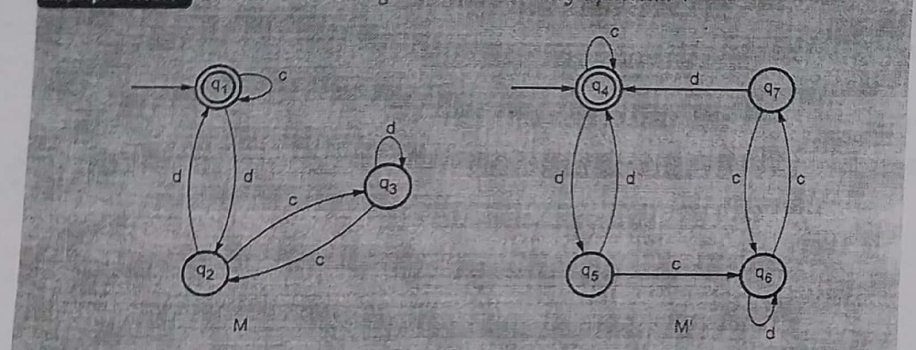


Fig. 1.13.1

Solution : We will first build the transition table for each input c and d. From first machine M on receiving input c in state q_1 we reach to state q_1 only. From second machine M', for state q_4 on receiving c we reach to state q_4. Thus for state (q_1, q_4), for input c we get next state as (q_1, q_4). Similarly for input d in state (q_1, q_4) we get next state as (q_2, q_5).

Both q_1 and q_4 are final states obtained in pair (q_1, q_4). Both q_2 and q_5 are nonfinal states obtained in pair (q_2, q_5) we will obtain transition for (q_2, q_5) for input c and d. The complete table is as given below.

	c	d
(q_1, q_4)	(q_1, q_4)	(q_2, q_5)
(q_2, q_5)	(q_3, q_6)	(q_1, q_4)
(q_3, q_6)	(q_2, q_7)	(q_3, q_6)
(q_2, q_7)	(q_3, q_6)	(q_1, q_4)

From the above table note that we do not get one final state and other nonfinal state in a pair. Hence we declare that two DFAs are equivalent.

Example 1.13.2 *Following are two FAs check whether they are equivalent or not.*

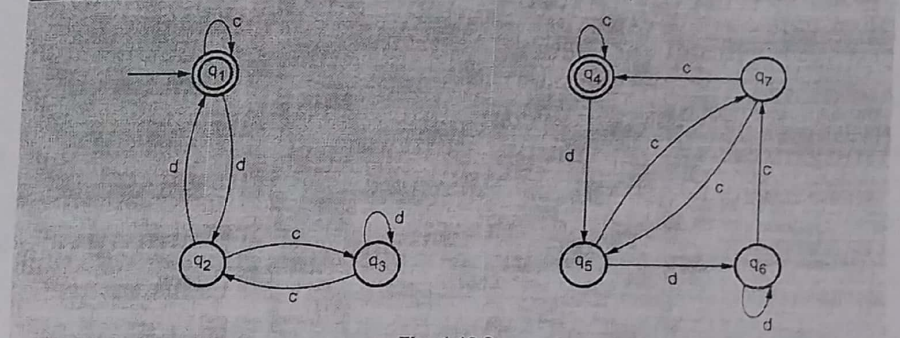


Fig. 1.13.2

Solution : We will design the transition table for each input symbol c and d as follows.

	c	d
(q_1, q_4)	(q_1, q_4)	(q_2, q_5)
(q_2, q_3)	(q_3, q_7)	(q_1, q_6)

We will terminate the construction of transition table because we get a pair (q_1, q_6) in which q_1 is a final state and q_6 is a nonfinal state. As per equivalence rule final and nonfinal state cannot form a pair. Hence the given FAs are not equivalent.

1.14 Applications of FA

Various applications of Finite Automata are -

1) In designing of lexical analysis phase of compiler, finite automata is used to recognize the tokens such as identifier, keywords, operators and so on.
2) The pattern of regular expression is recognized using finite automata.
3) It is used in text editors.
4) The finite automata is also used to design spell checkers.

Part III : Finite State Machine with Output

1.15 Moore and Mealy Machines

SPPU : Dec.-07,11,13,16,17, May-06,10,11, Aug.-17, Oct.-16,18, Marks 6

1.15.1 Definition and Construction

There are two types of FA with output and those are :

1) Moore machine 2) Mealy machine

Scanned with OKEN Scanner

Moore machine 1) o/p is the funcn of present state only

Moore machine is a finite state machine in which the next state is decided by current state and current input symbol. The output symbol at a given time depends only on the present state of the machine. The formal definition of Moore machine is,

Definition 2) more states 3) more logic is required to decode o/p, resulting in more circuit delays, react one clock edge later

Moore machine is a six tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where

- Q is finite set of states.
- Σ is finite set of input symbols.
- Δ is an output alphabet.
- δ is an transition function such that $Q \times \Sigma \rightarrow Q$. This is also known as state function.
- λ is output function $Q \rightarrow \Delta$. This function is also known as machine function.
- q_0 is the initial state of machine.

For example

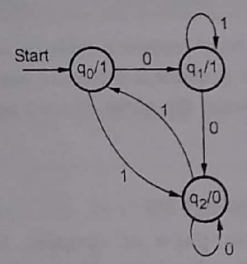


Fig. 1.15.1

Consider the Moore machine given below -

The transition table will be -

Current state	Next state (δ)		Output (λ)
	0	1	
q_0	q_1	q_2	1
q_1	q_2	q_1	1
q_2	q_2	q_0	0

In Moore machine output is associated with every state. In the above given Moore machine when machine is in q_0 state the output will be 1. For the Moore machine if the length of input string is n then output string has length n+1.

For the string 0110 then the output will be 11110.

Mealy machine 1) o/p is the funcn of present state as well as input

Mealy machine is a machine in which output symbol depends upon the present input symbol and present state of the machine. The Mealy machine can be defined as -

Definition 2) less states 3) reacts faster to input, generally in same clock cycle

Mealy machine is a six tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where

- Q is finite set of states.
- Σ is finite set of input symbols.
- Δ is an output alphabet.
- δ is state transition function such that $Q \times \Sigma \rightarrow Q$.
- λ is machine function such that $Q \times \Sigma \rightarrow \Delta$.
- q_0 is initial state of machine.

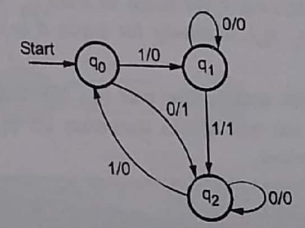


Fig. 1.15.2

For example,

For the input string 1001 the output will be 0001. In mealy machine the length of input string is equal to length of output string.

Difference between Moore and Mealy Machine :

1) Length of Moore machine is one longer than Mealy machine for given input.
2) Secondly output of the input will be along the edges in case of Mealy machine but it should be associated with the state in case of Moore machine.

difference is in the way the o/p is generated

Scanned with OKEN Scanner

Problems on Construction of Morre and Mealy Machine

Example 1.15.1 *Design a Moore machine to generate 1's complement of given binary number.*

SPPU : May-11, Marks 4

Solution :

- **Logic :** 1) To generate 1's complement of given binary number the simple logic which we will apply is that if input is 0 then output will be 1 and if input is 1 then output will be 0. 2) That means there are three state one-start state, second state is for taking 0's as input and produces output as 1. Then third state is for taking 1's as input and producing output as 0.

Hence the Moore machine will be,

- **Design**

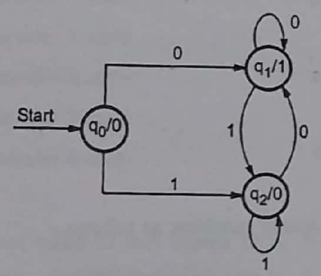


Fig. 1.15.3

- **Simulation**

For instance, take one binary number 1011 then

Input		1	0	1	1
State	q_0	q_2	q_1	q_2	q_2
Output	0	0	1	0	0

Thus we get 00100 as 1's complement of 1011, we can neglect the initial 0 and the output which we get is 0100 which is 1's complement of 1011. The transition table can be drawn as below -

Current state	Next state (δ) 0	Next state (δ) 1	Output (λ)
→ q_0	q_1	q_2	0
q_1	q_1	q_2	1
q_2	q_1	q_2	0

Thus Moore machine $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$; where $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0,1\}$, $\Delta = \{0,1\}$.

The transition table shows the δ and λ functions.

Example 1.15.2 *Design a Moore and Mealy machine for a binary input sequence such that if it has a substring 101 the machine outputs A if input has substring 110 it outputs B otherwise it outputs C.*

SPPU : Aug.-17, In Sem, Marks 6

Solution :

- **Logic :** For designing such a machine we need to take care of two conditions and those are checking 101 and checking 110. If we get 101 the output will be A. If we recognize 110 the output will be B. For other strings the output will be C. We can make a partial design of it as follows.

- **Design**

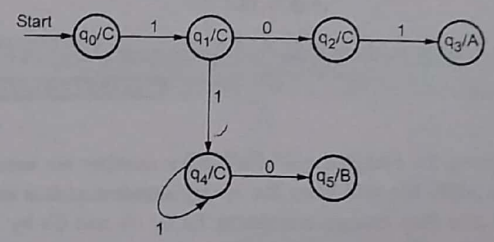


Fig. 1.15.4

Now we will insert the possibilities of 1's and 0's for each state. Then the Moore machine becomes.

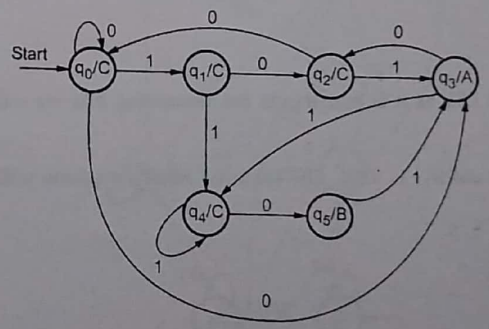


Fig. 1.15.5

Scanned with OKEN Scanner

Now the Mealy machine can be

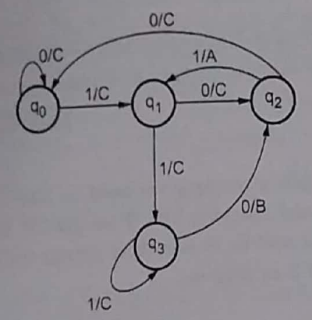


Fig. 1.15.6

Example 1.15.3 *Design a Mealy machine to find 2's complement of a given binary number.* **SPPU : Oct 18, In Sem, Marks 4**

Solution :

- **Logic :** For designing 2's complement of a binary number we assume that input is read from LSB to MSB. We will keep the binary number as it is until we read first 1. Keep that 1 as it is then change remaining 1's by 0's and 0's by 1's.

For example,

Let the binary number be

1011

←

read from LSB

Keep the first 1 from LSB as it is and toggle the remaining bits we will get

0101

Thus 2's complement of 1011 is 0101. The required Mealy machine will be -

- **Design**

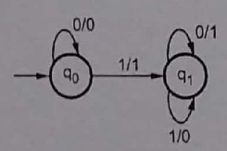


Fig. 1.15.7

Example 1.15.4 *Design a Moore machine which will increment the given binary number by 1.*

Solution : We will read the binary number form LSB one bit at a time. We will replace each 1 by 0 until we get first 0. Once we get first 0 we will replace it by 1 and then keep remaining bits as it is.

For example -

```
1  0  1  1   ← read from LSB bit by bit
1  0  1  0
            ↑
1  0  0  0
         ↑
1  1  0  0
   ↑
1  1  0  0
↑
```

Using this logic we can built a mealy machine as follows -

- **Design**

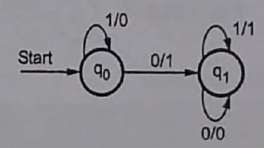


Fig. 1.15.8

Example 1.15.5 *Give Moore and Mealy machine for $\Sigma = \{0, 1, 2\}$, print the residue modulo 5 of input treated as a ternary number.* **SPPU : Dec.-07, May-10, Marks 6**

Solution :

- **Logic :** The ternary number is made up of 0,1 and 2. The interpretation of ternary number n can be -
  i) If we write 0 after n then number becomes 3n.
  ii) If we write 1 after n then number becomes 3n+1.
  iii) If we write 2 after n then number becomes 3n+2.

For example,

If n = 4 then its value is $4 \times 3^0 = 4$.

Scanned with OKEN Scanner

If we write 0 after 4 i.e.

$$40 = 4\times3^1+0\times3^0 = 12 \qquad \text{i.e. } (3\times4)$$

If we write 1 after 4 then

$$41 = 4\times3^1+1\times3^0 = 13 \qquad \text{i.e. } 3n+1$$

If we write 2 after 4 we get

$$42 = 4\times3^1+2\times3^0 = 14 \qquad \text{i.e. } 3n+2$$

For residue modulo 5 we will get remainder 0, remainder 1, remainder 2, remainder 3 and remainder 4 values. Then we assume various states for these remainders as -

q_0 - remainder 0 state

q_1 - remainder 1 state

q_2 - remainder 2 state

q_3 - remainder 3 state

q_4 - remainder 4 state

Now consider n = 4,

For 4.0 we get decimal value 12 that means 12 % 5 = 2 it gives remainder 2.

For 4.1 we get decimal value 13 that means 13 % 5 = 3 it gives remainder 3.

Similarly 4.2 given remainder 4.

n = 4 itself is remainder 4. Hence we can design,

- **Design**

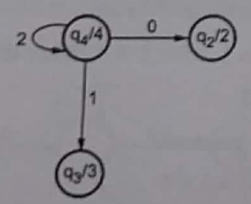


Fig. 1.15.9

Considering all possible cases we can design Moore machine as,

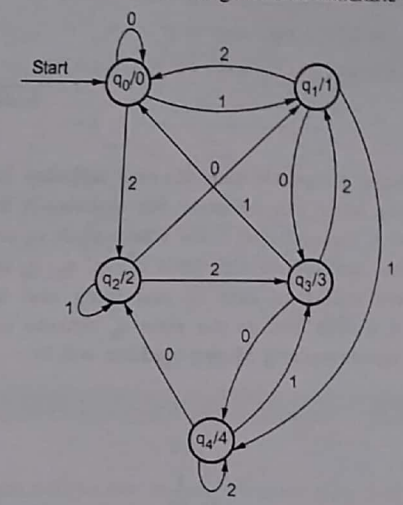


Fig. 1.15.10

Similarly the Mealy machine can be -

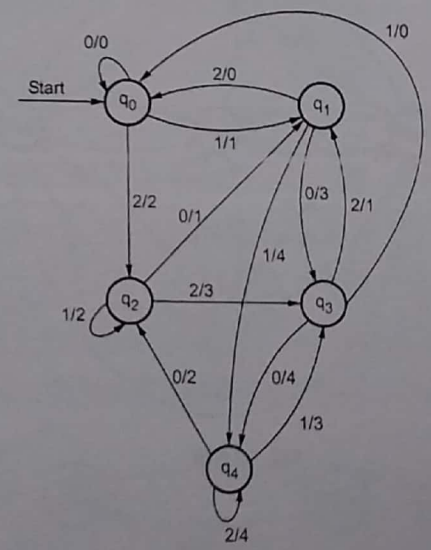


Fig. 1.15.11

Scanned with OKEN Scanner

Example 1.15.6 *Design a Moore machine that will read sequences made up of letters A, E, I, O, U and will give as output the same sequences except that in case where an I directly follows an E, it will be changed to U. Design the Mealy machine for the same.*

SPPU : May-06, Marks 8

Solution :

- **Logic :** We will assume a separate state for each alphabet. The output of that state will be corresponding letter. For instance For alphabet A the state will be q_0 and output of q_0 will be A. For alphabet E the state will be q_1 and output of q_1 will be E. Continuing in this fashion we will have q_0, q_1, q_2, q_3 and q_4 states. The start state will be q_0 we will take care of one thing and that is when I comes immediately after E it will lead to the state q_4 because output of state q_4 is u. With this logic the corresponding Moore machine will be -
- **Design**

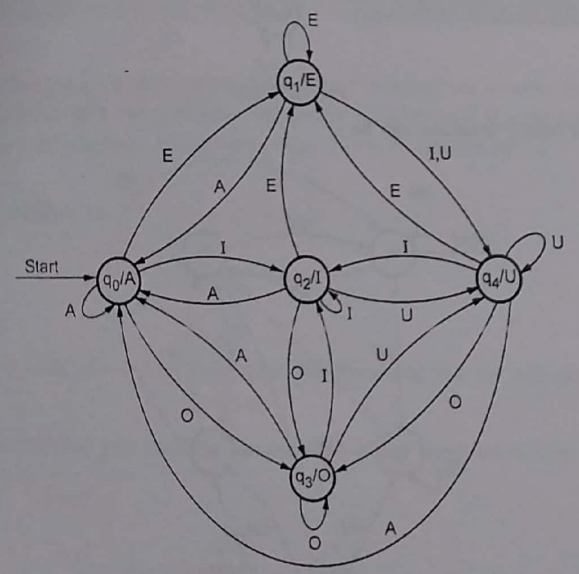


Fig. 1.15.12

Now we will draw the Mealy machine for the same problem.

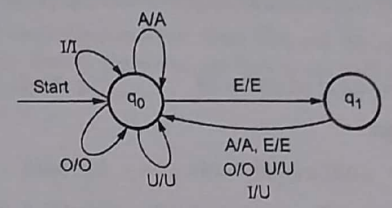


Fig. 1.15.13

Here the Mealy machine has the output along the edge itself we have got only two state. The transition changes from q_0 to q_1 when we get E. The I which is immediately following E will have the output U. Consider the string AIEOAEI will be give an output as AIEOAEU.

Example 1.15.7 *Construct a Moore machine to determine residue mod 3 for binary number.*

SPPU : Dec.-17, End Sem, Marks 6

Solution :

- **Logic :** This Moore machine is also called remainder 3 tester. In this machine we will get remainder 0, remainder 1 and remainder 2. To interpret the given binary number in its decimal value we consider n as a number if 0 is written after n then its value becomes 2n. If 1 is written after n then it's value becomes 2n + 1. For instance, if n = 0 then its decimal value is 0 then,

$$01 = 2n+1 = 1\times0+1 = 1$$

$$011 = 2n+1 = (1\times2)+1 = 3$$

consider n = 1 its decimal value is 1. After 1 if 0 comes then its value will be

$$10 = 2n = 2\times1 = 2$$

If 1 comes after 10 then its value becomes,

$$101 = 2n+1 = (2\times2)+1 = 5$$

With this logic we can construct a Moore machine with 3 states. q_0 is the start state and is considered as remainder 0 state. q_1 is considered to be remainder 1 state and q_2 is considered as remainder 2 state.

Scanned with OKEN Scanner

- **Design**

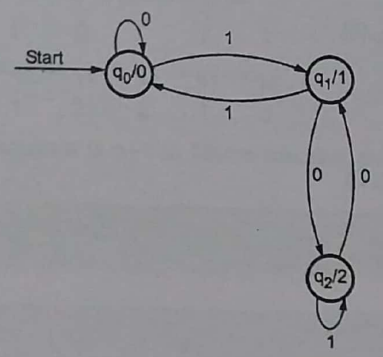
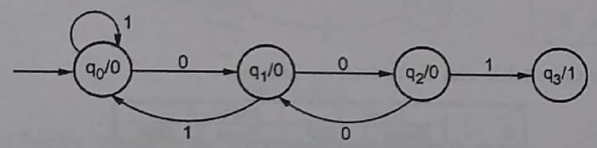


Fig. 1.15.14

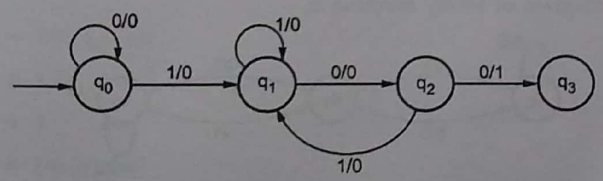
Example 1.15.8 *Design a Moore machine for generating output 1 if input of binary sequence 1 is preceded with exactly two zeroes.*

Solution : In this Moore machine on receiving input 001 the output will be 1. such a Moore machine can be shown as below -



Example 1.15.9 *Design a Mealy machine for a binary input sequence such that if the sequence ends with 100 the output is 1 otherwise output is 0.* **SPPU : May-10, Marks 6**

Solution : The Mealy machine will be -



Example 1.15.10 *Design a Moore machine for checking divisibility by 3 of a given decimal number (residue of 3).* **SPPU : Dec.-11, Marks 4**

Solution : The output at each state of Moore machine will be the remainder.

The Moore machine will be -

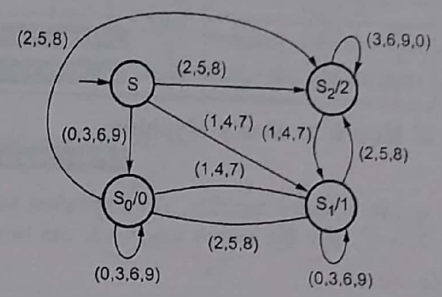


Fig. 1.15.15

Example 1.15.11 *Construct FSM for binary adder.* **SPPU : Dec.-13, Marks 6**

Solution :

In binary addition there will be two states - no carry state and a carry state.

The transition table will be.

x1	x2	Output
0	0	0
0	1	1
1	0	1
1	1	0

But carry gets generated

Present State	Next State, Output $x_1x_2 = 00$	01	10	11
No carry	No carry, 0	carry, 1	No carry, 1	carry, 0
carry	No carry, 1	carry, 0	carry, 0	carry, 1

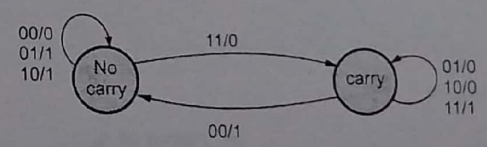


Fig. 1.15.16

Scanned with OKEN Scanner

University Questions

1. *Define and compare Moore and Mealy Machine.* **SPPU : Dec.-16 In Sem. Marks 4**
2. *Compare Moore and Mealy Machine with suitable example.* **SPPU : Oct.-16 In Sem, Marks 5**

1.16 Conversion of Moore to Mealy Machine

SPPU : May-11, Dec.-14, Marks 6

Let $M = (Q, \Sigma, \delta, \lambda, q_0)$ be a Moore machine. The equivalent Mealy machine can be represented by $M' = (Q, \Sigma, \delta, \lambda', q_0)$. The output function λ' can be obtained as -

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

Example 1.16.1 *The Moore machine to determine residue mod 3 for binary number is given below. Convert it to Mealy equivalent machine.*

Q \ Σ	0	1	Output (λ)
q_0	q_0	q_1	0
q_1	q_2	q_0	1
q_2	q_1	q_2	2

Solution : The transition diagram for the given problem can be drawn as -

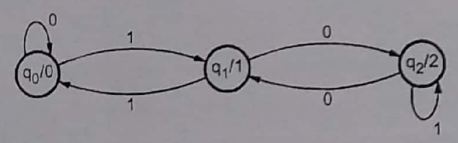


Fig. 1.16.1

The output function λ' can be obtained using following rule,

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

Hence we will obtain output for every transition corresponding to input symbol.

$$\lambda'(q_0, 0) = \lambda(\delta(q_0, 0))$$
$$= \lambda(q_0) \qquad \text{i.e. output of } q_0$$
$$\boxed{\lambda'(q_0, 0) = 0}$$
$$\lambda'(q_0, 1) = \lambda(\delta(q_0, 1))$$
$$= \lambda(q_1) \qquad \text{i.e. output of } q_1$$
$$\boxed{\lambda'(q_0, 1) = 1}$$

$$\lambda'(q_1, 0) = \lambda(\delta(q_1, 0))$$
$$= \lambda(q_2)$$
$$\boxed{\lambda'(q_1, 1) = 2}$$
$$\lambda'(q_1, 1) = \lambda(\delta(q_1, 1))$$
$$= \lambda(q_0)$$
$$\boxed{\lambda'(q_1, 1) = 0}$$
$$\lambda'(q_2, 0) = \lambda(\delta(q_2, 0))$$
$$= \lambda(q_1)$$
$$\boxed{\lambda'(q_2, 0) = 1}$$
$$\lambda'(q_2, 1) = \lambda(\delta(q_2, 1))$$
$$= \lambda(q_2)$$
$$\boxed{\lambda'(q_2, 1) = 2}$$

Hence the transition table can be drawn as follows -

Q \ Σ	Input 0		Input 1	
	State	O/P	State	O/P
q_0	q_0	0	q_1	1
q_1	q_2	2	q_0	0
q_2	q_1	1	q_2	2

The transition diagram of Mealy machine is,

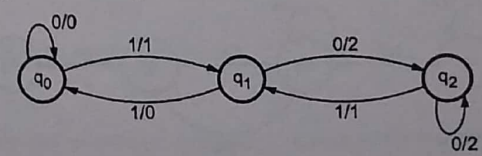


Fig. 1.16.2

The input string 10011 then the output for Mealy machine will be

next state	$q_1 \rightarrow$	$q_2 \rightarrow$	$q_2 \rightarrow$	$q_1 \rightarrow$	q_0
output	1	2	2	1	0

Scanned with OKEN Scanner

The output sequence for Moore machine will be

Input		1	0	0	1	1
next state	$q_0 \rightarrow$	$q_1 \rightarrow$	$q_2 \rightarrow$	$q_2 \rightarrow$	$q_1 \rightarrow$	q_0
output	0	1	2	2	1	0

The length of output sequence is n+1 in Moore machine and is n in Mealy machine which is desired.

Example 1.16.2 *Convert the following Moore machine into equivalent Mealy machine*
$M = (\{q_0, q_1\}, \{a, b\}, \{0,1\}, \delta, \lambda, q_0)$

Solution :

δ	a	b	Output (λ)
q_0	q_0	q_1	0
q_1	q_0	q_1	1

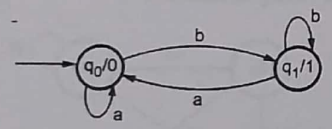


Fig. 1.16.3

The equivalent Mealy machine can be obtained as follows -

$$\lambda'(q_0, a) = \lambda(\delta(q_0, a)) = \lambda(q_0) = 0$$

$$\lambda'(q_0, b) = \lambda(\delta(q_0, b)) = \lambda(q_1) = 1$$

$$\lambda'(q_1, a) = \lambda(\delta(q_1, a)) = \lambda(q_0) = 0$$

$$\lambda'(q_1, b) = \lambda(\delta(q_1, b)) = \lambda(q_1) = 1$$

Hence the transition table can be drawn as follows -

Q \ Σ	Input a		Input b	
	State	O/P	State	O/P
q_0	q_0	0	q_1	1
q_1	q_0	0	q_1	1

The equivalent Mealy machine will be,

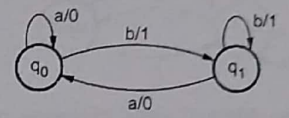


Fig. 1.16.4

Consider the output sequence for Moore machine for input sequence,

a b b a

Then

$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \xrightarrow{a} q_0$$

0 0 1 1 0

Similarly output sequence for Mealy machine will be

$$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \longrightarrow q_0$$

0 1 1 1 0

We can note that length of Moore machine is n+1 and that of Mealy machine is n.

Example 1.16.3 *Convert the following Moore machine to Mealy machine.*

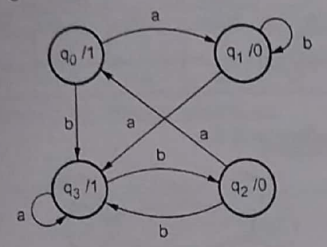
SPPU : May-11, Marks 6

State	Input		Output
	a	b	
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

Scanned with OKEN Scanner

Scanned with OKEN Scanner

Solution : The transition diagram for the given transition table will be



The output function λ' can be obtained using following rule,

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

$$\therefore \quad \lambda'(q_0, a) = \lambda(\delta(q_0, a))$$

$$= \lambda(q_1)$$

$$\lambda'(q_0, a) = 0$$

$$\lambda'(q_0, b) = \lambda(\delta(q_0, b))$$

$$= \lambda(q_3)$$

$$\lambda'(q_0, b) = 1$$

$$\lambda'(q_1, a) = \lambda(\delta(q_1, a))$$

$$= \lambda(q_3)$$

$$\lambda'(q_1, a) = 1$$

$$\lambda'(q_1, b) = \lambda(\delta(q_1, b))$$

$$= \lambda(q_1)$$

$$\lambda'(q_1, b) = 0$$

$$\lambda'(q_2, a) = \lambda(\delta(q_2, a))$$

$$= \lambda(q_0)$$

$$\lambda'(q_2, a) = 1$$

$$\lambda'(q_2, b) = \lambda(\delta(q_2, b))$$

$$= \lambda(q_3)$$

$$\lambda'(q_2, b) = 1$$

$$\lambda'(q_3, a) = \lambda(\delta(q_3, a)) = \lambda(q_3)$$

$$\lambda'(q_3, a) = 1$$

$$\lambda'(q_3, b) = \lambda(\delta(q_3, b))$$

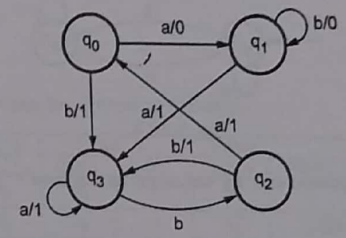
$$= \lambda(q_2)$$

$$\lambda'(q_3, b) = 0$$

The transition table can be as follows -

Q \ Σ	Input a		Input b	
	State	Output	State	Output
q_0	q_1	0	q_3	1
q_1	q_3	1	q_1	0
q_2	q_0	1	q_3	1
q_3	q_3	1	q_2	0

The transition diagram for Mealy machine is



Example 1.16.4 *Construct Mealy machine equivalent to the given Moore machine*

SPPU : Dec.-14 In Sem, Marks 6

	0	1	O/P
q_0	q_0	q_1	N
q_1	q_0	q_2	N
q_2	q_0	q_3	N
q_3	q_0	q_3	Y

Start state : q_0 ; Final state : q_3

Scanned with OKEN Scanner

Solution : The transition diagram for given Moore machine is shown below.

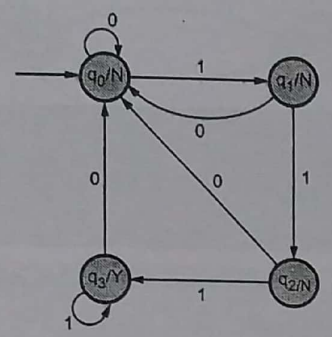


Fig. 1.16.5

The output function λ' can be obtained by following rule,

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

$$\therefore \quad \lambda'(q_0, 0) = \lambda(\delta(q_0, 0)) = \lambda(q_0)$$

$$\therefore \quad \boxed{\lambda'(q_0, 0) = N}$$

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

$$\therefore \quad \boxed{\lambda'(q_0, 1) = N}$$

$$\lambda'(q_1, 0) = \lambda(\delta(q_1, 0)) = \lambda(q_0)$$

$$\therefore \quad \boxed{\lambda'(q_1, 0) = N}$$

$$\lambda'(q_1, 1) = \lambda(\delta(q_1, 1)) = \lambda(q_2)$$

$$\therefore \quad \boxed{\lambda'(q_1, 1) = N}$$

$$\lambda'(q_2, 0) = \lambda(\delta(q_2, 0)) = \lambda(q_0)$$

$$\therefore \quad \boxed{\lambda'(q_2, 0) = N}$$

$$\lambda'(q_2, 1) = \lambda(\delta(q_2, 1)) = \lambda(q_3)$$

$$\therefore \quad \boxed{\lambda'(q_2, 1) = Y}$$

$$\lambda'(q_3, 0) = \lambda(\delta(q_3, 0)) = \lambda(q_0)$$

$$\therefore \quad \boxed{\lambda'(q_3, 0) = N}$$

$$\lambda'(q_3, 1) = \lambda(\delta(q_3, 1)) = \lambda(q_3)$$

$$\therefore \quad \boxed{\lambda'(q_3, 1) = Y}$$

Hence the transition table can be drawn as follows

Q \ Σ	Input 0		Input 1	
	State	O/P	State	O/P
q_0	q_0	N	q_1	N
q_1	q_0	N	q_2	N
q_2	q_0	N	q_3	Y
q_3	q_0	N	q_3	Y

The Mealy machine will be

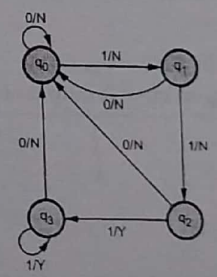


Fig. 1.16.6

Scanned with OKEN Scanner

1.17 Conversion of Mealy to Moore Machine

SPPU : May-13,18, Dec.-16, Marks 10

Method for conversion of Mealy Machine to Moore Machine

Let $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ be a Mealy machine then there exists a Moore machine M' equivalent to M.

The M' can be given as $M' = (Q \times \Delta, \Sigma, \Delta, \delta', \lambda', [q_0, b_0])$ where b_0 is an output symbol selected from Δ. We will calculate δ' and λ' as follows -

$$\delta'([q,b], a) = [\delta(q,a), \lambda(q,a)]$$

$$\lambda'([q,b]) = b$$

δ' defines the move made by M' on input a.

M' on input a.

λ' defines the output made for the corresponding state q.

Thus we have to calculate δ' and λ' for $q_0, q_1, q_2, \ldots\ldots q_n$ on input $a_0, a_1, \ldots\ldots a_n$ and should emit the outputs $b_1, b_2, b_3, \ldots\ldots b_n$.

Example 1.17.1 *Convert the following Mealy machine into equivalent Moore machine.*

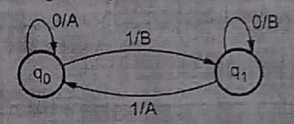


Fig. 1.17.1

Solution : The states for Moore machine will be $[q_0, A], [q_0, B], [q_1, A], [q_1, B]$. Then we will calculate δ' and λ' as follows -

$$\delta'((q_0, A), 0) = [\delta(q_0, 0), \lambda(q_0, 0)]$$

i.e [next state of q_0 on input 0, output of q_0 on input 0]

q_0 A

$$= [q_0, A]$$

$\lambda'([q_0, A]) = A \rightarrow$ i.e. output of state q_0.

$$\delta'([q_1, A], 0) = [\delta(q_1, 0), \lambda(q_1, 0)]$$

$$= [q_1, B]$$

$$\lambda'([q_1, A]) = A$$

$$\delta'([q_1, A], 1) = [\delta(q_1, 1), \lambda(q_1, 1)]$$

$$= [q_0, A]$$

$$\lambda'([q_1, A]) = A$$

Hence,

	0	1	Output
$[q_0, A]$	$[q_0, A]$	$[q_1, B]$	A
$[q_1, A]$	$[q_1, B]$	$[q_0, A]$	A

Continuing in this fashion we can calculate δ' and λ' as follows -

$$\delta'([q_0, B], 0) = [\delta(q_0, 0), \lambda(q_0, 0)]$$

$$= [q_0, A]$$

$$\lambda'[q_0, B]) = B$$

$$\delta'((q_1, B), 1) = [\delta(q_0, 1), \lambda(q_0, 1)]$$

$$= [q_1, B]$$

$$\lambda'([q_0, B]) = B$$

$$\delta'([q_1, B], 0) = [\delta(q_1, 0), \lambda(q_1, 0)]$$

$$= [q_1, B]$$

$$\lambda'([q_1, B]) = B$$

$$\delta'((q_1, B), 1) = [\delta(q_1, 1), \lambda(q_1, 1)]$$

$$= [q_0, B]$$

$$\lambda'([q_1, B]) = B$$

Thus we get the transition table as

State \ I/P	0	1	Output
$[q_0, A]$	$[q_0, A]$	$[q_1, B]$	A
$[q_0, B]$	$[q_0, A]$	$[q_1, B]$	B
$[q_1, A]$	$[q_1, B]$	$[q_0, A]$	A
$[q_1, B]$	$[q_1, B]$	$[q_0, A]$	B

Scanned with OKEN Scanner

The transition diagram will be,

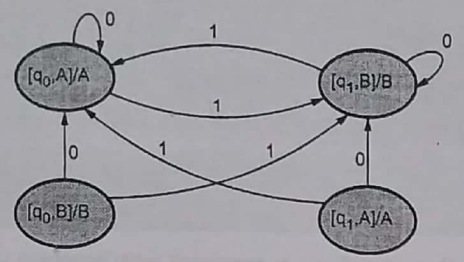


Fig. 1.17.2

Example 1.17.2 *Convert the following Mealy machine into equivalent Moore machine.*

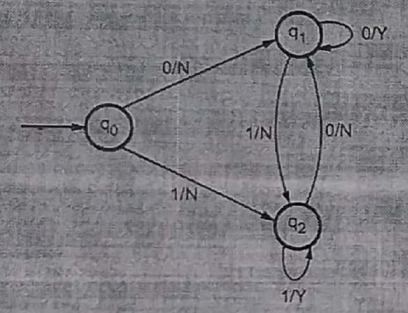


Fig. 1.17.3

Solution : We will write the transition table for given transition graph -

	Input 0	Output	Input 1	Output
q_0	q_1	N	q_2	N
q_1	q_1	Y	q_2	N
q_2	q_1	N	q_2	Y

Now we will find out the states and corresponding outputs for Moore machine. The states for Moore machine will be -

$[q_0,N]$, $[q_0,Y]$, $[q_1,N]$, $[q_1,Y]$, $[q_2,N]$, $[q_2,Y]$

Now let us calculate δ' and λ' for all the above given states -

$$\delta'([q_0,N)], 0) = [\delta(q_0,0), \lambda(q_0,0)]$$
$$= [q_1, N]$$

$$\lambda'([q_0,N]) = N$$

Similarly,

$$\delta'([q_0,N), 1) = [\delta(q_0,1), \lambda(q_0,1)]$$
$$= [q_2, N]$$
$$\lambda'([q_0,N]) = N$$

The partial transition table is -

State \ I/P	0	1	Output
$[q_0, N]$	$[q_1, N]$	$[q_2, N]$	N

Now, for remaining states the corresponding transitions and outputs can be obtained as follows -

$$\delta'([q_0,Y], 0) = [\delta(q_0,0), \lambda(q_0,0)]$$
$$= [q_1, N]$$
$$\lambda'([q_0,Y]) = Y$$
$$\delta'([q_0,Y], 1) = [\delta(q_0,1), \lambda(q_0,1)]$$
$$= [q_2 N]$$
$$\lambda'([q_0,Y]) = Y$$
$$\delta'([q_1N], 0) = [\delta(q_1,0), \lambda(q_1,0)]$$
$$= [q_1 Y]$$
$$\lambda'([q_1,N]) = N$$
$$\delta'([q_1,N], 1) = [\delta(q_1,1), \lambda(q_1,1)]$$
$$= [q_2, N]$$
$$\lambda'([q_1,N]) = N$$
$$\delta'([q_1,Y], 0) = [\delta(q_1,0), \lambda(q_1,0)]$$
$$= [q_1, Y]$$
$$\lambda'([q_1,Y]) = Y$$
$$\delta'([q_2,N], 0) = [\delta(q_2,0), \lambda(q_2,0)]$$
$$= [q_1, N]$$
$$\lambda'([q_2,N]) = N$$
$$\delta'([q_2,N], 1) = [\delta(q_2,1), \lambda(q_2,1)]$$

Scanned with OKEN Scanner

$= [q_2, Y]$

$\lambda'([q_2, N]) = N$

$\delta'([q_1, Y], 1) = [\delta(q_1, 1), \lambda(q_1, 1)]$

$= [q_2, N]$

$\lambda'([q_1, Y]) = Y$

$\delta'([q_2, Y], 0) = [\delta(q_2, 0), \lambda(q_2, 0)]$

$= [q_1, N]$

$\lambda'([q_2, Y]) = Y$

$\delta'([q_2, Y], 1) = [\delta(q_2, 1), \lambda(q_2, 1)]$

$= [q_2, Y]$

$\lambda'([q_2, Y]) = Y$

The transition table can be drawn as -

State \ Σ	0	1	Output
$[q_0, N]$	$[q_1, N]$	$[q_2, N]$	N
$[q_0, Y]$	$[q_1, N]$	$[q_2, N]$	Y
$[q_1, N]$	$[q_1, Y]$	$[q_2, N]$	N
$[q_1, Y]$	$[q_1, Y]$	$[q_2, N]$	Y
$[q_2, N]$	$[q_1, N]$	$[q_2, Y]$	N
$[q_2, Y]$	$[q_1, N]$	$[q_2, Y]$	Y

Example 1.17.3 *Convert the following Mealy machine into equivalent Moore machine.*

SPPU : May-13, Marks 10

Solution : We will design transition table for Mealy machine.

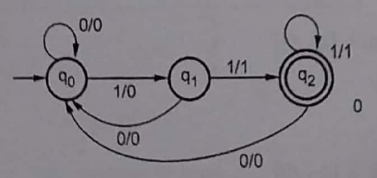


Fig. 1.17.4

Current state	A = 0		A = 1	
	Next state	Output	Next state	Output
q_0	q_0	0	q_1	0
q_1	q_0	0	q_2	1
q_2	q_0	0	q_2	1

q_0 is associated with output 0, q_1 is associated with output 0, q_2 is associated with output 1

Moore machine

Current state	Next state		Output
	A = 0	A = 1	
q_0	q_0	q_1	0
q_1	q_0	q_2	0
q_2	q_0	q_2	1

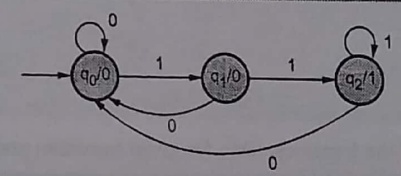
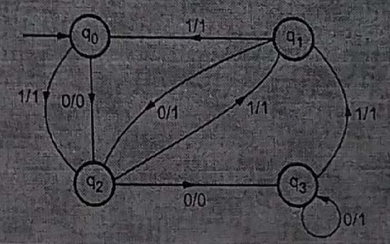


Fig. 1.17.5

Example 1.17.4 *Construct Moore machine equivalent for the given Mealy machine.*



Scanned with OKEN Scanner

Solution : The states for Moore machine will be $[q_0, 0]$, $[q_0, 1]$, $[q_1, 0]$, $[q_1, 1]$, $[q_2, 0]$, $[q_2, 1]$, $[q_3, 0]$, $[q_3, 1]$.

Then we will calculate δ' and λ' functions.

$$\delta'([q_0, 0], 0) = [\delta(q_0, 0), \lambda(q_0, 0)] = [q_2, 0]$$
$$\lambda'([q_0, 0]) = 0$$
$$\delta'([q_0, 0], 1) = [\delta(q_0, 1), \lambda(q_0, 1)] = [q_2, 1]$$
$$\lambda'([q_0, 0]) = 0$$

The partial transition table will be

State \ Input	0	1	Output
$[q_0, 0]$	$[q_2, 0]$	$[q_2, 1]$	0

Now remaining states for corresponding transitions are as given below -

$$\delta'([q_0, 1], 0) = [\delta(q_0, 0), \lambda(q_0, 0)] = [q_2, 0]$$
$$\lambda'([q_0, 1]) = 1$$
$$\delta'([q_0, 1], 1) = [\delta(q_0, 1), \lambda(q_0, 1)] = [q_2, 1]$$
$$\lambda'([q_0, 1]) = 1$$
$$\delta'([q_1, 0], 0) = [\delta(q_1, 0), \lambda(q_1, 0)] = [q_2, 1]$$
$$\lambda'(q_1, 0) = 0$$
$$\delta'([q_1, 0], 1) = (\delta(q_1, 1), \lambda(q_1, 1)) = [q_0, 1]$$
$$\delta'([q_1, 0]) = 0$$
$$\delta'([q_1, 1], 0) = (\delta(q_1, 0), \lambda(q_1, 0)) = [q_2, 1]$$
$$\delta'([q_1, 1]) = 1$$
$$\delta'([q_1, 1], 1) = (\delta(q_1, 1), \lambda(q_1, 1)) = [q_0, 1]$$
$$\lambda'([q_1, 1]) = 1$$
$$\delta'([q_2, 0], 0) = (\delta(q_2, 0), \lambda(q_2, 0)) = [q_3, 0]$$
$$\lambda'([q_2, 0]) = 0$$

$$\delta'([q_2, 0], 1) = (\delta(q_2, 1), \lambda(q_2, 1)) = [q_1, 1]$$
$$\lambda'([q_2, 0]) = 0$$
$$\delta'([q_2, 1], 0) = (\delta(q_2, 0), \lambda(q_2, 0)) = [q_3, 0]$$
$$\lambda'([q_2, 1]) = 1$$
$$\delta'([q_2, 1], 1) = (\delta(q_2, 1), \lambda(q_2, 1)) = [q_1, 1]$$
$$\lambda'([q_2, 1]) = 1$$
$$\delta'([q_3, 0], 0) = (\delta(q_3, 0), \lambda(q_3, 0)] = [q_3, 1]$$
$$\lambda'([q_3, 0]) = 0$$
$$\delta'([q_3, 0], 1) = (\delta(q_3, 1), \lambda(q_3, 1)) = [q_1, 1]$$
$$\lambda'([q_3, 0]) = 0$$
$$\delta'([q_3, 1], 0) = (\delta(q_3, 0), \lambda(q_3, 0)) = [q_3, 1]$$
$$\lambda'([q_3, 1]) = 1$$
$$\delta'([q_3, 1], 1) = (\delta(q_3, 1), \lambda(q_3, 1)) = [q_1, 1]$$
$$\lambda'([q_3, 1]) = 1$$

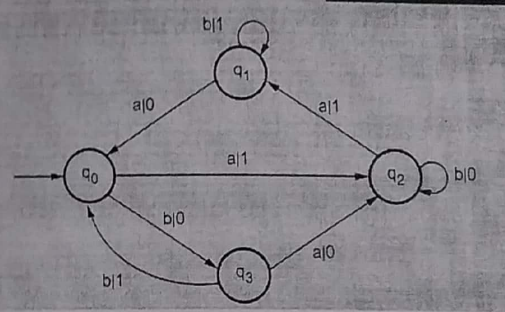
The transition table for Moore machine will be

State \ Input	0	1	Output
$[q_0, 0]$	$[q_2, 0]$	$[q_2, 1]$	0
$[q_0, 1]$	$[q_2, 0]$	$[q_2, 1]$	1
$[q_1, 0]$	$[q_2, 1]$	$[q_0, 1]$	0
$[q_1, 1]$	$[q_2, 1]$	$[q_0, 1]$	1
$[q_2, 0]$	$[q_3, 0]$	$[q_1, 1]$	0
$[q_2, 1]$	$[q_3, 0]$	$[q_1, 1]$	1
$[q_3, 0]$	$[q_3, 1]$	$[q_1, 1]$	0
$[q_3, 1]$	$[q_3, 1]$	$[q_1, 1]$	1

Scanned with OKEN Scanner

Example 1.17.5 *Construct Moore machine for given Mealy machine.*

SPPU : Dec.-16 End Sem, Marks 6



Solution : We will write the transition table for given transition graph

State \ Transit	Input a	Output	Input b	Output
q_0	q_2	1	q_3	0
q_1	q_0	0	q_1	1
q_2	q_1	1	q_2	0
q_3	q_2	0	q_0	1

Now we will find out the states and corresponding outputs for Moore machine. The states for Moore machine will be $[q_0,1]$, $[q_0,0]$, $[q_1,0]$, $[q_1,1]$, $[q_2,0]$, $[q_2,1]$, $[q_3,0]$, $[q_3,1]$

Now let us compute λ' and δ' for the above states.

$$\delta'([q_0,1],a) = [\delta(q_0,a),\lambda(q_0,a)]$$
$$= [q_2,1]$$

Similarly

$$\lambda'([q_0,1]) = 1$$
$$\delta'([q_0,1],b) = [\delta(q_0,b),\lambda(q_0,b)]$$
$$= [q_3,0]$$
$$\delta'([q_0,0],a) = [\delta(q_0,a),\lambda(q_0,a)] = [q_2,1]$$
$$\lambda'([q_0,0]) = 0$$
$$\delta'([q_0,0],b) = [\delta(q_0,b),\lambda(q_0,b)]$$

$$= [q_3,0]$$
$$\lambda'([q_0,0]) = 0$$
$$\delta'([q_1,0],a) = [\delta(q_1,a),\lambda(q_1,a)] = [q_0,0]$$
$$\delta'([q_1,0]) = 0$$
$$\delta'([q_1,0],b) = [\delta(q_1,b),\lambda(q_1,b)] = [q_1,1]$$
$$\delta'([q_1,1],a) = [\delta(q_1,a),\lambda(q_1,a)] = [q_0,0]$$
$$\lambda([q_1,1]) = 1$$
$$\delta'([q_1,1],b) = [\delta(q_1,b),\lambda(q_1,b)] = [q_1,1]$$
$$\delta'([q_2,0],a) = [\delta(q_2,a),\lambda(q_2,a)] = [q_1,1]$$
$$\lambda([q_2,0]) = 0$$
$$\delta'([q_2,0],b) = [\delta(q_2,b),\lambda(q_2,b)] = [q_2,0]$$
$$\delta'([q_2,1],a) = [\delta(q_2,a),\lambda(q_2,a)] = [q_1,1]$$
$$\lambda([q_2,1]) = 1$$
$$\delta'([q_3,0],a) = [\delta(q_3,a),\lambda(q_3,a)] = [q_2,0]$$
$$\lambda([q_3,0]) = 0$$
$$\delta'([q_3,0],b) = [\delta(q_3,b),\lambda(q_3,b)] = [q_0,1]$$
$$\delta'([q_3,1],a) = [\delta(q_3,a),\lambda(q_3,a)] = [q_2,0]$$
$$\lambda([q_3,1]) = 1$$
$$\delta'([q_3,1],b) = [\delta(q_3,b),\lambda(q_3,b)] = [q_0,1]$$
$$\lambda([q_3,1]) = 1$$

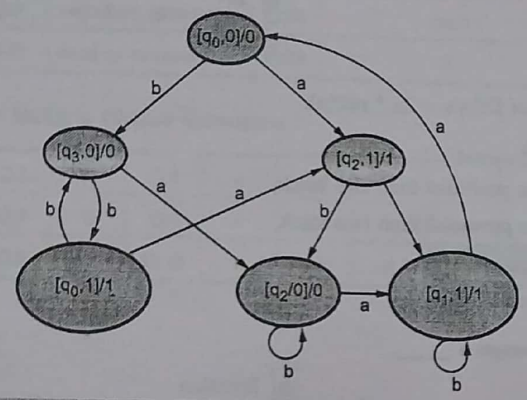
Thus we have obtained next state and output. The transition table can be drawn as follows :

State \ Σ	a	b	Output
$[q_0,0]$	$[q_2,1]$	$[q_3,0]$	0
$[q_0,1]$	$[q_2,1]$	$[q_3,0]$	1
$[q_1,0]$	$[q_0,0]$	$[q_1,1]$	0
$[q_1,1]$	$[q_0,0]$	$[q_1,1]$	1

Scanned with OKEN Scanner

$[q_2, 0]$	$[q_1, 1]$	$[q_2, 0]$	0
$[q_2, 1]$	$[q_1, 1]$	$[q_2, 0]$	1
$[q_3, 0]$	$[q_2, 0]$	$[q_0, 1]$	0
$[q_3, 1]$	$[q_2, 0]$	$[q_0, 1]$	1

The transition diagram for Moore machine will be



Example 1.17.6 *Construct the Mealy machine to accept strings ending with '00' or '11' over $\Sigma = \{0, 1\}$. Convert Mealy machine into equivalent Moore machine.*

SPPU : May-18, Marks 8

Solution : The Mealy machine is

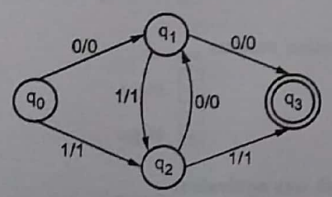


Fig. 1.17.6

The conversion of Mealy to Moore - We will write transition table as,

	i/p 0	Output	i/p 1	Output
q_0	q_1	0	q_2	1
q_1	q_3	0	q_2	1
q_2	q_1	0	q_3	1

Now we will obtain states and corresponding outputs for Moore machine. The states for Moore machine will be

$[q_0, 0], [q_0, 1], [q_1, 0], [q_1, 1]$

We will calculate δ' and λ' for all above states

$$\delta'([q_0, 0], 0) = [\delta(q_0, 0)\lambda(q_0, 0)]$$
$$= [q_1, 0]$$
$$\lambda'[(q_0, 0)] = 0$$

Similarly,

$$\delta'([q_0, 0], 1) = [\delta(q_0, 1)\lambda(q_0, 1)]$$
$$= [q_2, 1]$$
$$\lambda'[(q_0, 0)] = 0$$

The partial transistion table

State \ i/p	0	1	Output
$[q_0, 0]$	$[q_1, 0]$	$[q_2, 1]$	0

Thus, if we compute δ' and λ' in above manner we get following Moore machine.

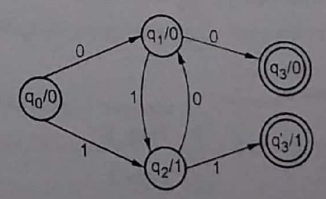


Fig. 1.17.7

Scanned with OKEN Scanner

Answer Keys for Fill in the Blanks :

Q.1	Deterministic Finite Automate	Q.2	regular language
Q.3	same	Q.4	ε or ∧
Q.5	non regular	Q.6	regular language
Q.7	Yes	Q.8	Same
Q.9	cognitive, generative	Q.10	can't
Q.11	dead or unreachable	Q.12	6

Answer Keys for Multiple Choice Questions :

Q.1	d	Q.2	c	Q.3	d	Q.4	a
Q.5	d	Q.6	c	Q.7	b	Q.8	c
Q.9	a	Q.10	c	Q.11	d		

❑❑❑

UNIT- II

2 Regular Expressions and Languages

Syllabus

Regular Expressions (RE) : *Definition and Identities of RE, Operators of RE, Equivalence of two regular expressions, Equivalence of regular expressions and regular languages (RL), Conversion of RE to FA using direct method, Conversion of FA to RE using Arden's theorem, Pumping lemma for RLs, Closure properties of RLs, Applications of Regular Expressions.*

Contents

2.1 Definition of Regular Expressions (RE) ..... **May-15,** ..... Marks 2

2.2 Identities of RE and Equivalence of Two Regular Expressions
..... **Aug.-17, Dec.-09,12,**
..... **May-14,** ..... Marks 8

2.3 Construction of Regular Expression **May-09,10,13,15,16,**
..... **Dec.-04,12,14,18,19, Aug.-15,**
..... **Oct.-18,19,** ..... Marks 8

2.4 Conversion of RE to FA **Dec.-10,11,12,16,19, May-11,16,**
..... **Oct.-18, Aug.-17,** ..... Marks 8

2.5 Conversion of FA to RE using Arden's Theorem
..... **Dec.-10, May-11,14,15,19,**
..... **Aug.-15,17, Oct.-16,18,** .. Marks 8

2.6 Pumping Lemma for Regular Language **May-09,11,16,18,19, Oct.-16,18,**
..... **Dec.-08,12,14,16,19,** .... Marks 6

2.7 Closure Properties of Regular Language

2.8 Applications of Regular Expressions ..... **Oct.-16, Dec.-19,** ..... Marks 5

2.9 Fill in the Blanks

2.10 Multiple Choice Questions

Scanned with OKEN Scanner

language accepted by FA

2.1 Definition of Regular Expressions (RE)

SPPU : May-15, Marks 2

represents regular language

- Let Σ be an alphabet which is used to denote the input set. The regular expression over Σ can be defined as follows.

Definition :

1. ϕ is a regular expression which denotes the **empty set**.
2. ε is a regular expression and denotes the set $\{\varepsilon\}$ and it is a null string.
3. For each 'a' in Σ 'a' is a regular expression and denotes the set $\{a\}$.
4. If r and s are regular expressions denoting the languages L_1 and L_2 respectively, then
   r+s is equivalent to $L_1 \cup L_2$ i.e. union.
   rs is equivalent to $L_1 L_2$ i.e. concatenation.
   r^* is equivalent to L_1^* i.e. closure.
   The r^* is known as **kleen closure** or closure which indicates occurrence of r for ∞ number of times.

Collection of null strings

- For example if $\Sigma = \{a\}$ and we have regular expression $R = a^*$, then R is a set denoted by $\rightarrow a^*$
  $R = \{\varepsilon, a, aa, aaa, aaaa, ...\}$ That is R includes any number of a's as well as empty string which indicates zero number of a's appearing, denoted by ε character.
- Similarly there is a **positive closure** of L which can be shown as L^+. The L^+ denotes set of all the strings except the ε or null string. The null string can be denoted by ε or ^
- If $\Sigma = \{a\}$ and if we have regular expression $R = a^+$ then R is a set denoted by
  $R = \{a, aa, aaa, aaaa, ...\}$

We can construct L^* as

$$L^* = \varepsilon \cdot L^+$$

University Question

1. *With examples define regular expression.* **SPPU : May-15, (End Sem), Marks 2**

2.2 Identities of RE and Equivalence of Two Regular Expressions

SPPU : Aug.-17, Dec.-09,12, May-14, Marks 8

- The two regular expressions P and Q are equivalent (denoted as P = Q) if and only if P represents the same set of strings as Q does. For showing this

equivalence of regular expressions we need to show some identities of regular expressions.

- Let P, Q and R are regular expressions then the identity rules are as given below.

1.	$\varepsilon R = R\varepsilon = R$	
2.	$\varepsilon^* = \varepsilon$	ε is null string
3.	$(\phi)^* = \varepsilon$	ϕ is empty string.
4.	$\phi R = R\phi = \phi$	
5.	$\phi + R = R$	
6.	$R + R = R$	
7.	$RR^* = R^*R = R^+$	
8.	$(R^*)^* = R^*$	
9.	$\varepsilon + RR^* = R^*$	
10.	$(P + Q) R = PR + QR$	
11.	$(P + Q)^* = (P^* Q^*) = (P^* + Q^*)^*$	
12.	$R^* (\varepsilon + R) = (\varepsilon + R) R^* = R^*$	
13.	$(R + \varepsilon)^* = R^*$	
14.	$\varepsilon + R^* = R^*$	
15.	$(PQ)^* P = P (QP)^*$	
16.	$R^* R + R = R^* R$	

$(a+b)^* = \{\varepsilon, a, b, ab, aa, ab \in\}$

collection of all strings with a & b

Example 2.2.1 *Prove* $(1+00^*1)+(1+00^*1)(0+10^*1)^*(0+10^*1) = 0^*1(0+10^*1)^*$

Solution : Let us solve L.H.S. first,

$$(1+00^*1)+(1+00^*1)\,(0+10^*1)^*\,(0+10^*1)$$

We will take $(1+00^*1)$ as a common factor

Scanned with OKEN Scanner

$1+00^*1 \quad \underbrace{(\varepsilon+(0+10^*1)^*(0+10^*1))}$

$(\varepsilon+R^*R)$ where $R = (0+10^*1)$

As we know, $(\varepsilon + R^*R) = (\varepsilon + RR^*) = R^*$

$\therefore (1+00^*1)\ ((0+10^*1)^*)$ out of this consider

$\underbrace{(1+00^*1)}\ (0+10^*1)^*$

Taking 1 as a common factor

$(\varepsilon+00^*)\ 1\ (0+10^*1)^*$

Applying $\varepsilon+00^* = 0^*$

$0^*1\ (0+10^*1)^*$

= R.H.S.

Hence the two regular expressions are equivalent.

Example 2.2.2 *Prove* $\varepsilon + 1^*(011)^*(1^*(011)^*)^* = (1+011)^*$ **SPPU : Aug.-17, In Sem, Marks 4**

Solution : Let, L.H.S. is

$\varepsilon + \underbrace{1^*(011)^*}\ \underbrace{(1^*(011)^*)^*}$

If we consider $1^*(011)^*$ as P_1 then,

$= \varepsilon + P_1 P_1^*$

$= P_1^*$ $\qquad \because \varepsilon + P_1P_1^* = P_1^*$

We can put $P_1 = 1^*(011)^*$ then,

$= (1^*(011)^*)^*$

Now consider $P_2 = 1$ and $P_3 = (011)$ then it becomes

$= (P_2^*\ P_3^*)^*$

$= (P_2 + P_3)^*$ $\qquad \because (P^*Q^*)^* = (P+Q)^*$

$= (1+011)^*$

= R.H.S.

Thus L.H.S. = R.H.S.

Hence $\varepsilon + 1^*(011)^*(1^*(011)^*)^* = (1+011)^*$ is proved.

Example 2.2.3 *Show that* $(0^*1^*)^* = (0+1)^*$ **SPPU : Dec.-09, Marks 4**

Solution :

L.H.S = $(0^*1^*)^*$ = {ε, 0, 00, 000, 1, 11, 111, 01, 10, 001, ...}

R.H.S = $(0+1)^*$ = {ε, 0, 00, 000, 1, 11, 111, 01, 10, 001, ...}

That means both the regular expressions allow any combination of 0's and 1's

As L.H.S = R.H.S.

$(0^*1^*)^* = (0+1)^*$ is proved.

Example 2.2.4 *Show that - i)* $R^*R = R^+$ *ii)* $(P+Q)^* = (P^*Q^*)^*$ **SPPU : Dec.-12, Marks 6**

Solution : i) R is relation set

So R^* = {∈, R, RR, RRR .....}

$R^*R = R^*\cdot R$ = {∈, R, RR, RRR, ...}R = {R, RR, RRR, RRRR ....}

$= R^+$

Hence proved.

ii) Consider R.H.S. $= (P^*Q^*)^*$

= {∈, P, Q, PP, QQ, PQ, QP ...}

= {any combination of P's, any combination of Q's or any combination of P or Q or ∈}

$= (P+Q)^*$

Hence proved.

iii) Consider $(R^*)^*$

R is relation set

So $(R^*)^*$ = {∈, R, RR, RRR, ...}

$(R)^*$ = {∈, R, RR, RRR, ...}

So both expression gives same set of string hence $(R^*)^* = (R)^*$.

Scanned with OKEN Scanner

string) ~~aaa~~ of length at least ⇒ L = { aa, ab, ba, bb, aaa ... }

R = (a+b)(a+b)(a+b)*

Example 2.2.5 *Prove that :*

i) $(111^*)^* = (11+111)^*$ ii) $(0^*1^*)^* = (0+1)^*$ **SPPU : May-14, Marks 8**

Solution :

i) L.H.S. = $(111^*)^* = \{\varepsilon, 11, 111, 1111, 11111, \ldots\}$

R.H.S. = $(11+111)^* = \{\varepsilon, 11, 111, 1111, 11111, \ldots\}$

As L.H.S. = R.H.S.

$(111)^* = (11 + 111)^*$.

ii) $(0^*1^*)^* = \{\varepsilon, 0, 1, 00, 11, 001, 10, 100, \ldots\}$

$(0^*1^*)^* = (0^*+1^*)^*$

$(0+1)^* = \{\varepsilon, 0, 1, 00, 11, 001, 10, 100, \ldots\}$

This shows that

$(0^*1^*)^* = (0+1)^*$

2.3 Construction of Regular Expression

SPPU : May-09,10,13,15,16, Dec.-04,12,14,18,19, Aug.-15, Oct.-18,19, Marks 8

The construction of Regular Expression is possible using the concatenation, kleen closure and positive closure operators. Various regular expressions can be understood with the help of following examples

Problems based on Regular Expressions

Example 2.3.1 *Write the regular expression for the language accepting all combinations of a's over the set $\Sigma = \{a\}$.*

Solution : All combinations of *a*'s means *a* may be single, double, tripple and so on. There may be the case that *a* is appearing for zero times, which means a null string. That is we expect the set of $\{\varepsilon, a, aa, aaa, \ldots\}$. So we can give regular expression for this as

$$R = a^*$$

That is kleen closure of *a*.

Example 2.3.2 *Design the regular expression (r.e.) for the language accepting all combinations of a's except the null string over $\Sigma = \{a\}$.*

Solution : The regular expression has to be built for the language

$$L = \{a, aa, aaa, \ldots\}$$

This set indicates that there is no null string. So we can denote r.e. as

$$R = a^+$$

As we know, positive closure indicates the set of strings without a null string.

Example 2.3.3 *Design regular expression for the language containing all the strings containing any number of a's and b's.*

Solution : The regular expression will be

$$\text{r.e.} = (a + b)^*$$

This will give the set as L = {ε, *a*, *aa*, *ab*, b, b*a*, b*ab*, *abab*, ... any combination of *a* and b}.

The $(a + b)^*$ means any combination with *a* and b even a null string.

Example 2.3.4 *Construct the regular expression for the language containing all strings having any number of a's and b's, except the null string.*

Solution : r.e. = $(a+b)^+$

This regular expression will give the set of strings of any combination of a's and b's except a null string.

Example 2.3.5 *Construct the r.e. for the language accepting all the strings which are ending with 00 over the set $\Sigma = \{0, 1\}$.*

Solution : The r.e. has to be formed in which at the end, there should be 00. That means

r.e. = (any combination of 0's and 1's) 00

i.e. r.e. = $(0+1)^*\,00$

Thus the valid string are 100, 0100, 1000, 10100 .... strings ending with 00.

Example 2.3.6 *Write r.e. for the language accepting the strings which are starting with 1 and ending with 0, over the set $\Sigma = \{0,1\}$.*

Solution : The first symbol in r.e. should be 1 and the last symbol should be 0.

So, $R = 1(0+1)^*0$

Note that the condition is strictly followed by keeping starting and ending symbols correctly. In between them there can be any combination of 0 and 1 including a null string.

Example 2.3.7 *If L = {The language starting and ending with a and having any combination of b's in between }, then what is r ?*

Scanned with OKEN Scanner

Solution : The regular expression

$$r = a\, b^*\, a$$

Example 2.3.8 *Describe in simple English the language represented by the following regular expression $r = (a + ab)^*$*

Solution : We will first try to find out the set of strings, which can be possible by this r,

$$L(r) = \{a, aba, abab, aab, aaa, ...\}$$

The language is beginning with a but not having consecutive (in a clump) b's.

Example 2.3.9 *Write regular expression to denote the language L over Σ^*, where $\Sigma = \{a, b, c\}$ in which every string will be such that any number of a's is followed by any number of b's is followed by any number of c's.*

Solution : As we have seen any number of a's means a^* any number of b's means b^* any number of c's means c^*. Since as given in problem statement, b's appear after a's and c's appear after b's. So the regular expression could be -

$$r = a^*b^*c$$

Example 2.3.10 *Write a regular expression to denote a language L over Σ^*, where $\Sigma = \{a, b, c\}$ such that every string will have atleast one a followed by atleast one b followed by atleast one c.*

Solution : Now, in this problem the condition is slightly changed to "atleast". That means the null string is not allowed at all. So, we can write

$$R = a^+b^+c^+$$

Example 2.3.11 *Write r.e. to denote a language L which accepts all the strings which begin or end with either 00 or 11.*

Solution : The r.e. can be categorized into two subparts.

$$R = L_1 + L_2$$

L_1 = The strings which begin with 00 or 11.

L_2 = The strings which end with 00 or 11.

Let us find out L_1 and L_2.

$$L_1 = (00 + 11)\ (\text{any number of 0's and 1's})$$

$$L_1 = (00 + 11)\ (0+1)^*$$

Similarly,

$$L_2 = (\text{any number of 0's and 1's})\ (00 + 11)$$

$$= (0 + 1)^*\ (00 + 11)$$

Hence

$$R = [(00+11)(0+1)^*] + [(0+1)^*(00+11)]$$

Example 2.3.12 *Write a r.e. to denote a language L over Σ^*, where $\Sigma = \{a, b\}$ such that the 3rd character from right end of the string is always a.*

Solution : The r.e. contains the third symbol from right end as a

r = [any number of character of a's and b's] [a] [either a or b] [either a or b]

3rd 2nd 1st

$$r = (a+b)^*\ a\ (a+b)\ (a+b)$$

Thus the valid strings are b*a*bb or b*aaa* or *baba* or *aaa*b and so on.

Example 2.3.13 *Construct r.e. for the language L which accepts all the strings with atleast two b's over the $\Sigma = \{a, b\}$.*

Solution : $R = (a + b)^*\ b\ (a + b)^*\ b\ (a + b)^*$

Atleast two b's are maintained but the two b's are surrounded by any combination of a's and b's.

Example 2.3.14 *Construct r.e. for the language which consists of exactly two b's over the set $\Sigma = \{a, b\}$.*

Solution : Now, this problem is similar to the previous problem. But only difference is that there should be exactly two b's.

$$R = a^*\ b\ a^*\ b\ a^*$$

a^* indicates either the string contains any number of a's or a null string. Thus we can derive any string having exactly two b's and any number of a's.

Example 2.3.15 *Write r.e. which contains L having strings which should have atleast one 0 and atleast one 1.*

Solution : The required expression will be

$$R = [(0+1)^*0\ (0+1)^*1(0+1)^*] + [(0+1)^*1(0+1)^*0(0+1)^*]$$

Scanned with OKEN Scanner

Example 2.3.16 *Construct r.e. which denotes a language L over the set $\Sigma = \{0\}$ having even length of string.*

Solution : Since $\Sigma = \{0\}$ there are strings in L of even length i.e.

L = {ε, 00, 0000, 000000, ...}.

So we can give the regular expression as

$$R = (00)^*$$

Thus kleen closure indicates the recursion of two zeros which will ultimately give the even length of the string.

Example 2.3.17 *Write r.e. which denotes a language L over the set $\Sigma = \{1\}$ having odd length of strings.*

Solution : $R = 1\ (11)^*$

The odd length strings are L = {1, 111, 11111, ...}. Note the difference between previous example and this one.

Example 2.3.18 *Describe the language denoted by following regular expression*

r.e. $= (b^*(aaa)^*b^*)^*$

Solution : The language can be predicted from the r.e. by finding out the meaning of it. We can split the r.e. as

r.e. = (any combination of b's) $(aaa)^*$ (any combination of b's).

L = {The language consists of the strings in which *a*'s appear trippled, there is no restriction on number of b's}

Example 2.3.19 *Construct regular expression for the language L over the set $\Sigma = \{a, b\}$ in which the total number of a's are divisible by 3.*

Solution : The total number of *a*'s are divisible by 3. So the valid strings can be

L = {*baaa*, *bababa*, *ababab*, ...}

The regular expression can be

$$\text{r.e.} = (b^*\, a\, b^*\, a\, b^*\, a\, b^*)^*$$

Example 2.3.20 *Write the r.e. to denote the language L over $\Sigma = \{a, b\}$ such that all the strings do not contain the substring "ab".*

Solution : The L = {ε, *a*, b, bb, *aa*, b*a*, b*aa* ...}

$$\text{The r.e.} = (b^*\, a^*)$$

In this regular expression if we substitute $a^* = \varepsilon$ we get all combination of b's and similarly if we substitute $b^* = \varepsilon$ then we will get all combinations of *a*'s. We have strictly maintained a condition for not to have *ab* as substring by giving the regular expression as $b^*\, a^*$.

Example 2.3.21 *Find all possible regular expression $L \subseteq \{a, b\}^*$ for*

a) The set of all strings ending in b

b) The set of all strings ending in ba

c) The set of all strings ending neither b nor ba.

Solution :

a) The regular expression for language containing strings ending with b is

$$\text{r.e.} = (a + b)^*\, b.$$

b) The regular expression for language containing strings ending with ba is

$$\text{r.e.} = (a + b)^*\, ba$$

c) The regular expression for language containing strings ending with neither b nor a is

$$\text{r.e.} = (a + b)^*\, ab + a^*$$

Example 2.3.22 *Let L be a language. It is clear from the definition that $L^+ \subseteq L^*$. Under what circumstances are they equal ?*

Solution :

The language L^+ denotes any combination of input symbols without ε.

The language L^* denotes any combination of input symbols along with ε.

Hence $L^+ \subseteq L^*$

But if $L_1 = \varepsilon + L^+$ then L_1 becomes equal to L^*. That means if ε is added as one symbol in the set generated by L^+ then L^* become equal.

Example 2.3.23 *Find all possible regular expression $L \subseteq \{a, b\}^*$ for*

a) The set of all strings ending in ab

b) The set of all strings ending in ba

c) The set of all stings ending neither ab nor ba.

Solution :

a) The regular expression is

$$\text{r.e.} = (a + b)^*\, ab$$

Scanned with OKEN Scanner

b) The regular expression is

r.e. = $(a + b)^* ba$

c) The regular expression is

$(a + b)^* aa + (a + b)^* bb$

Example 2.3.24 *Consider the two regular expressions*
$r = 0^* + 1^*$, $s = 01^* + 10^* + 1^* 0 + (0^* 1)^*$.
a) Find the string corresponding to r but not to s
b) Find the string corresponding to s but not to r
c) Find the string corresponding to both r and f.

Solution : a) The string corresponding to r but not to s : {00,000,0000, ...}

b) The string corresponding to s but not to r : {01 or 10}

c) The string corresponding to both r and f : {0, 1, 11, 111, 1111, 11111, ...}

Example 2.3.25 *Find a regular expression corresponding to each of the following subsets of {0, 1}**
1) The language of all strings containing exactly two 0's.
2) The language of all strings containing at least two 0's.
3) The expression of all strings that do not end with 01.

Solution :

1) If a language contains all strings containing exactly two 0's then, it can be at any position, may or may not be surrounded by 1's. Hence r.e. will be

r.e. = $1^* 0 1^* 01^*$

2) If a language contains all strings containing at least two 0's, then it may be at any position, and may or may not be surrounded by $(0+1)^*$. Hence r.e. will be
r.e. = $(0+1)^* 0 (0+1)^* 0 (0+1)^*$

3) The language contains all the strings that do not end with 01 means it may end with either 0's or end with either 1's or end with 10. Hence r.e. will be
- r.e. = $[(0+1)^* 0^*] + [(0+1)^* (10)^*]$

Example 2.3.26 *Obtain the regular expressions for the following sets :*
1) $L1 = \{b^2, b^5, b^8, b^{11}, b^{14} ...\}$ 2) $L2 = \{a^{2n+1} | n > 0\}$

Solution : 1) $L1 = \{b^2, b^5, b^8, b^{11}, b^{14} ...\}$

The regular expression for $L1 = \{b^2, b^5, b^8, b^{11}, ...\}$ is

Regular expression = $bb (bbb)^*$

2) $L2 = \{a^{2n+1} | n > 0\}$

The regular expression for $L2 = \{a^{2n+1} | n > 0\}$ is

Regular expression = $a (aa)^+$

Example 2.3.27 *Obtain in plain english language represented by following regular expression.*

a) $0^*(10^*10^*)^*1(0^*10^*1)^*0^*$ *b)* $0^*(0^*10^*1)^*0^*$

Solution : a) $0^* (10^*10^*)^* 1 (0^*10^*1)^* 0^*$:

L = {The language containing group of even number of 1's separated by single 1}

b) $0^* (0^* 10^* 1)^* 0^*$:

L = {The language containing even number of 1's and any number of 0's}

Example 2.3.28 *Represent following formal languages using regular expressions.*
1) All string's of a's and b's without any combination of double letters.
2) All string's of 0's and 1's with even number of 0's.
3) All string's of a's and b's containing at least two a's.

Solution : 1) All string's of a's and b's without any combination of double letters : $(01)^* + (10)^*$

2) All string's of 0's and 1's with even number of 0's : $(1^* 01^* 01^*)^*$

3) All string's of a's and b's containing at least two a's : $(b^* a b^* a b^*)^+$

Example 2.3.29 *Find the regular expressions representing the following sets :*
i) The set of all strings over {a, b} with three consecutive b's.
ii) The set of all strings over {0, 1} beginning with 00.
iii) The set of all strings over {0, 1} ending with 00 and beginning with 1.

Solution : i) r.e. = $a^* (bbb)^* a^*$

ii) r.e. = $00 (0+1)^*$

iii) r.e. = $1 (0+1)^* 00$

Example 2.3.30 *Find regular expressions representing the following sets :*
i) The set of all strings over {a, b} having atmost one pair of a's or atmost one pair of b's.
ii) The set of all strings over {a, b} in which the number of occurrences of a is divisible by
iii) The set of all strings over {a, b} in which there are at least two occurrences of b between any two occurrences of a.

Scanned with OKEN Scanner

Solution : i) The set of all strings over {a, b} having atmost one pair of a's or atmost one pair of b's.

$((ab)+b)^* a a ((ba)+b)^* / ((ba)+a)^* bb ((ab)+a)^*$

ii) The set of all strings over {a, b} in which the number of occurance of a is divisible by 3.

$\Rightarrow \quad b^* (aaa)^* b^*$

iii) The set of all strings over {a, b}, in which these are at least two occurances of b between any two occurances of a.

$\Rightarrow \quad (a+b)^* (aa) (bb)^+ (aa) (a+b)^*$

Example 2.3.31 *Write R.E. for the following*

i) Σ = (0, 1) odd number of 1's in strings

ii) Σ = (0, 1) triple 0 must never appear in string.

Solution : i) r.e. = (0 * 10 *) (10 * 10 *) *

ii) r.e. = (1 + 01) * 00 (1 + 10) *

Example 2.3.32 *Give english description of the language of the following regular expression*

$(1 + \varepsilon) (00^*1)^* 0^*$

Solution : This expression generates the strings as -

{ ε, 1, 101, 1001, 10, 100, 1010, ...}

This regular expression gnerates all strings in which every 1 is separated by one or more 0. It also accepts the string which has only one 1 or the empty string.

Example 2.3.33 *Write regular expressions for the following languages over $\{0, 1\}^*$*

i) The set of strings that begin with 110

ii) The set of all strings not containing 101 as a substring.

Solution : i) r.e. = 110 (0 + 1) *

ii) r.e. = (0 + 11 * 00) * (ε + 11 * (ε + 0))

Example 2.3.34 *Describe in English the language represented by following regular expressions :*

i) $(a + ab)^*$ ii) $(a + b)^* a (a + b)^*$ iii) $(a^* ab^* ab^*) + b^*$ iv) $a^+ b^* c^+$.

SPPU : May-09, Marks 4

Solution :

i) L = The language containing the strings in which each 'b' is preceded by 'a', and every string is beginning with 'a'.

ii) L = The language containing the strings which contain at least one 'a'.

iii) L = The language containing the strings of either any number of b's or the strings containing at least two a's.

iv) L = The language containing the string which contain at least one 'a' followed by any number of b's followed by at least one 'c'.

Example 2.3.35 *Write a regular expression for the following :*

i) Σ = {a, b} such that each of string do not have aa or bb as a substring in it.

ii) Σ = {0, 1} with even number of 0's.

iii) Σ = {a, b} such that ab is not a substring of any strings. **SPPU : May-09, Marks 6**

Solution : i) r.e. = (ab)* + (ba)* + (aba)* + (bab)*

ii) r.e. = (00)*

iii) r.e. = (b*a*).

Example 2.3.36 *Represent the following using regular expressions*

i) Σ = {a, b, c} the language such that "any number of a's followed by any number of c'S"

ii) If L(r) = {0, 2, 01, 21, 211, 0111, ...} then what is r ?

iii) If L(r) = {00, 010, 0110, 01110, ...} then what is r ?

iv) Language defined over Σ = {a, b} has to have the strings beginning with 'a' and not to have two consecutive a's that is the regular expression for the same.

SPPU : Dec.-04, Marks 4

Solution :

i) r.e = $(a^* b^* + b^*)^*$

ii) r.e. = $(01^* + 21^*)$

iii) r.e. = $01^* 0$

iv) r.e. = $a (b+ba)^*$

Example 2.3.37 *Find a regular expression corresponding to each of the following subsets of $\{0, 1\}^*$.*

i) The language of all strings ending in 01.

ii) The language of all strings that contain at least one occurrence of each symbol in Σ.

iii) The language of all string containing an even no of 0's. **SPPU : May-10, Marks 6**

Solution :

i) r.e. = $(0+1)^* 01$

ii) r.e. = $0 (0+1)^* 1 + 1 (0+1)^* 0$

iii) r.e. = $(1^* 01^* 01^*)^* + 1^*$

Scanned with OKEN Scanner

Example 2.3.38 *Give regular expression for the following :*

i) For $\sum = \{0, 1\}$ *such that*

$L(r) = \{w \in \sum^* : w$ *has at least one pair of consecutive zeroes*$\}$

ii) For $\sum = \{0, 1\}$ *such that*

$L(r) = \{w \in \{0,1\}^* : w$ *has no pair of consecutive zeroes*$\}$

iii) For $\sum = \{0, 1\}$ *such that*

$L(r) = \{a^n, b^m, n \geq 4, m \leq 3\}$

iv) For $\sum = \{0, 1\}$ *such that*

$L = \{$*all strings containing even numbers of zeros*$\}$. **SPPU : May-10, Marks 8**

Solution :

i) r.e. = $(0+1)^* 00(0+1)^*$

ii) r.e. = $(0+\varepsilon)(1+10)^*$

iii) r.e. = $(aaaa)^+ b(\varepsilon+b+bb)$

iv) r.e. = $(1^*01^*01^*)^* + 1^*$

Example 2.3.39 *Give RE for the following languages over* $\sum = \{0,1\}$.

i) Strings containing even number of 1's followed by odd number of 0's.

ii) Strings that do not contain three consecutive 0's.

iii) Strings that contain at most three 0's. **SPPU : Dec.-12, Marks 6**

Solution :

i) $(00)^*(1(11)^*)$

ii) $(1)^*((01+001))^*(1)^*$ or $1^*(01+001)^*1^*$

iii) $(1)^*0(1)^*0(1)^*0(1)^*$ or $1^*01^*01^*01^*$

Example 2.3.40 *Find all strings of length 5 or less in the regular set represented by the following -*

i) $(ab+a)^*(aa+b)$ *ii)* $(a^*b+b^*a)^*a$ *iii)* $a^*+(ab+a)^*$ **SPPU : Dec.-12, Marks 6**

Solution :

i) {b, a, ab, aa, aaa, aab, abb, abab, abaa, abab, aaaa, aaab, aaba, aaaaa, aaaab, aabab, ababa, ababb, abaaa, aabab}

ii) {ba, aa, aba, baa, aaba, bbaa, aaaba, bbbaa}

iii) {a, aa, ab, aaa, aab, aba, aaaa, abab, aaba, aaab, abaa, aaaaa, ababa, abaaa, aaaab, aabab, abaab}.

Example 2.3.41 *Find all possible regular expression ever* $L \subseteq \{0,1\}^*$.

i) The set of all possible string containing exactly two 0's.

ii) The set of all string that do not end with "01". **SPPU : May-13, Marks 4**

Solution : i) r.e. = $1^*01^*01^*$

ii) $(\varepsilon+0+1+(0+1)^*(00+10+11))$

Example 2.3.42 *Define regular expressions. Give RE for the following over* $\Sigma = \{0, 1\}$

i) All binary strings with at least one 0.

ii) All binary strings with at most one 0. **SPPU : Dec.-14, In Sem, Marks 6**

Solution : i) $(0+1)^* 0(0+1)^*$ ii) $1^* 0 1^*$

Example 2.3.43 *Let* $\sum = \{a,b\}$. *Write RE to define language consisting of strings such that*

i) Strings without substring bb

ii) Strings that have exactly one double letter in them. **SPPU : May-15, In Sem, Marks 4**

Solution : i) r.e. = $(a+ba)^*(b+\varepsilon)$

ii) r.e. = $[(b+\varepsilon)(ab)^* aa(ba)^*(b+\varepsilon)]+[(a+\varepsilon)(ba)^* bb(ab)^*(a+\varepsilon)]$

Example 2.3.44 *Find the regular expression corresponding to each of the following subset of {0, 1}.*

i) Language of all Strings not containing the substring 000.

ii) Language of all Strings containing an even no of 0's.

SPPU : Aug.-15, In Sem, Marks 4

Solution : i) r.e. = $(1+01+001)(\varepsilon+0+00)$

ii) $(1^*0 \;\; 1^*0 \; 1^*)^* + 1^*$

Example 2.3.45 *Describe in simple English the language defined by the following RE*

i) $(a+b)^* a(a+b)^*$

ii) $(01^*0)1$

iii) $a(a+b)^* bb$ **SPPU : May-16, End Sem, Marks 6**

Solution :

i) The language contains at least singe a.

ii) The language contains the strings made up of Zero's and One's which contains the substring which ends in 1 and in between two Zero's there are any number of 1's.

iii) The language made up of a's and b's in which the substring begins with single a and ends with double b.

Scanned with OKEN Scanner

Example 2.3.46 *Write regular language for the following regular expressions.*

i) $r_1 = (0+1)^* \cdot 11(0+1)^*$ *ii)* $r_2 = (1+10)^*$ **SPPU : Oct.-18, In Sem, Marks 4**

Solution : i) L = {The language contains at least one pair of 1}

ii) L = {The language does not contain consecutive 0's}

Example 2.3.47 *Find the regular expression for the language*

i) Consisting of all strings of a's and b's without any combination of double letters.

ii) Over Σ : = {a, b} containing at least one 'a' and at least one 'b'.

iii) Consisting of set of all strings that start with 'a' and do not have two consecutive 'b's. **SPPU : Dec.-18, End Sem, Marks 6**

Solution : i) r.e. = $(ab)^* + (ba)^*$

ii) r.e. = $b^* a b^* b a^*$

iii) r.e. = $a^*(ba)^*$

Example 2.3.48 *Determine the regular expressions over the Σ = {a,b} for the following*

i) Set of all strings containing exactly 2 a's.

ii) Set of all strings containing atleast 2 a's.

iii) Set of all strings that do not consist of two consecutive 0's

SPPU : Oct.-19, In Sem, Marks 6

Solution : i) r.e. = $b^* ab^* ab$

ii) r.e. = $(a+b)^* a(a+b)^* a(a+b)^*$

iii) r.e. = $(b+ab)^* (a+\varepsilon)$

If the input set Σ = {0,1} then the r·e = $(1+01)^* (0+\varepsilon)$

Example 2.3.49 *Define formal definition of RE. Also give the regular expression for the following languages :*

i) The set of strings over the alphabet {a,b} starting with b and ending with odd numbers of a's or even numbers of b's.

ii) The set {10, 1010}

iii) If L(r) = {ε, x, xx, xxx, xxxx, xxxxx} what is r ? **SPPU : Dec.-19, End Sem, Marks 6**

Solution : Regular Expression : Refer section 2.1

i) r.e. = $b(a+b)^*[a(aa)^* + (bb)^*]$

ii) r.e. = $(10+1010)$

iii) $r = x^*$

Example 2.3.50 *Describe in simple English the language defined by the following regular expressions :*

i) $(a+b)^* \cdot aa \cdot (a+b)^*$ *ii)* $a+b^* \cdot c+\varepsilon$ **SPPU : Oct.-19, In Sem, Marks 4**

Solution : i) Let, r.e. = $(a+b)^* aa(a+b)^*$

This regular language contains atleast one double a.

ii) r.e. = $a+b^*c+\varepsilon$

The language contains {ε, a, c, bc, bbc, bbbc,...}

2.4 Conversion of RE to FA

SPPU : Dec.-10,11,12,16,19, May-11,16, Oct.-18, Aug.-17, Marks 8

Theorem 1 : Let r be a regular expression, then there exists a NFA with ε transitions that accepts L (r).

Proof : This theorem can be proved by induction method.

The basis of induction will be by considering r has zero operators.

Basis (zero operators) - Now, since r has zero operators, means r can be either ε or ϕ or *a* for some *a* in input set Σ .

The finite automata for the same can be written as

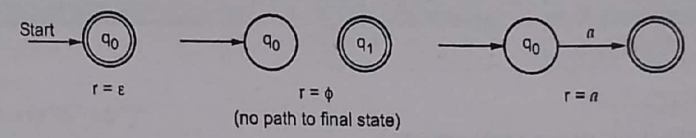


Fig. 2.4.1 Finite automata for given regular expression

Induction : This theorem can be true for n number of operators. The n is greater than or equal to 1. The regular expression contains equal to or more than one operators.

In any type of regular expression there are only three cases possible.

1. Union 2. Concatenation 3. Closure.

Let us see each,

Case 1 : Union case

- Let $r = r_1 + r_2$ where r_1 and r_2 be the regular expressions.

There exists two NFA's $M_1 = (Q_1, \Sigma_1, \delta_1, \{f_1\})$

and $M_2 = (Q_2, \Sigma_2, \delta_2, \{f_2\})$

- $L(M_1) = L(r_1)$ means the language states by regular expression r_1 is same which is represented by M_1. Similarly $L(M_2) = L(r_2)$.

Scanned with OKEN Scanner

Q_1 represents the set of all the states in machine M_1.

Q_2 represents the set of all the states in machine M_2.

- We assume that Q_1 and Q_2 are totally different i.e. Q_1 and Q_2 are disjoint.
- Let q_0 be new initial state and f_0 be the new final state we will form

$$M = ((Q_1 \cup Q_2 \cup \{q_0, f_0\}), (\Sigma_1 \cup \Sigma_2), \delta, q_0 \{f_0\})$$

- The δ is denoted by,

i) $\delta(q_0, \varepsilon) = \{q_1, q_2\}$

ii) $\delta(q, a) = \delta_1(q, a)$ for q in $Q_1 - \{f_1\}$ and a in $\Sigma_1 \cup \{\varepsilon\}$.

iii) $\delta(q, a) = \delta_2(q, a)$ for q in $Q_2 - \{f_2\}$ and a in $\Sigma_2 \cup \{\varepsilon\}$.

iv) $\delta(f_1, \varepsilon) = \delta_1(f_2, \varepsilon) = \{f_0\}$

- All the moves are now present in machine M which is as shown Fig. 2.4.2.

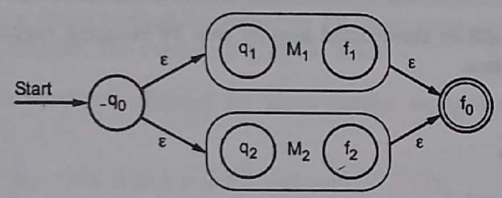


Fig. 2.4.2 The machine M for union

- The construction of machine M is shows the transition from q_0 to f_0 must begin by going to q_1 or q_2 on ε. If the path goes to q_1, then it follows the path in machine M_1 and goes to the state f_1 and then to f_0 on ε.
- Similarly, if the path goes to q_2, then it follows the path in machine M_2 and goes to state f_2 and then to f_0 on ε.
- Thus the L (M) = L $(M_1) \cup L(M_2)$. That means either the path in machine M_1 or M_2 will be followed. This define L(M) to be union of L (M_1) and L (M_2).

Case 2 : Concatenation case

- Consider that there are two regular expressions r_1 and r_2 such that $r = r_1 r_2$. The M_1 and M_2 denotes the two machines such that L (M_1) = L (r_1) and L (M_2) = L (r_2).
- The construction of machine M will be

$$M = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta, \{q_1\}, \{f_2\})$$

- The mapping function δ will be given as

i) $\delta(q, a) = \delta_1(q, a)$ for q in $Q_1 - \{f_1\}$ and a in $\Sigma_1 \cup \{\varepsilon\}$

ii) $\delta(f_1, \varepsilon) = \{q_2\}$

iii) $\delta(q, a) = \delta_2(q, a)$ for q in Q_2 and a in $\Sigma_1 \cup \{\varepsilon\}$

The machine M is shown in the Fig. 2.4.3.

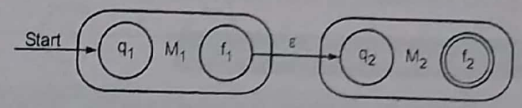


Fig. 2.4.3 Machine M for concatenation

- The initial state is q_1 by some input a the next state will be f_1. And on receiving ε the transition will be from f_1 to q_2 and the final state will be f_2. The transition from q_2 to f_2 will be on receiving some input b.
- Thus L (M) = ab

That is a is in L (M_1) and b is in L (M_2).

- Hence we can prove L (M) = L (M_1) L (M_2).

Case 3 : Closure case

- Let $r = r_1^*$ where r_1 be a regular expression.
- The machine M_1 is such that $L(M_1) \neq L(r_1)$.
- Then construct $M = (Q_1 \cup \{q_0, f_0\}, \Sigma_1, \delta, q_0, \{f_0\})$
- The mapping function δ is given by,

i) $\delta(q_0, \varepsilon) = \delta(f_1, \varepsilon) = \{q_1, f_0\}$

ii) $\delta(q, a) = \delta_1(q, a)$ for q in $Q_1 - \{f_1\}$ and a in $\Sigma_1 \cup \{\varepsilon\}$

- The machine M will be

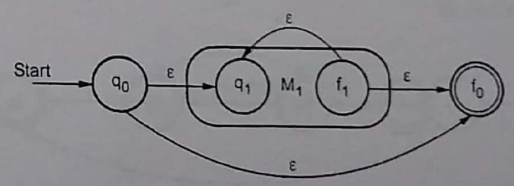


Fig. 2.4.4

- The machine M_1 shows that from q_0 to q_1 there is a transition on receiving ε similarly, from q_0 to f_0 on ε there is a path.
- The path exists from f_1 to q_1, a back path. Similarly a transition from f_1 to f_0 final state, on receiving ε. The total recursion is possible.
- Thus one can derive ε, a, aa, aaa, ... for the input a.

Thus L (M) = L $(M_1)^*$ is proved.

Scanned with OKEN Scanner

Now based on this proof let us solve some examples. These examples illustrate how to convert given regular expression to NFA with ε moves.

Problems based on Construction of NFA from r.e.

Example 2.4.1 *Construct NFA for the regular expression $b+ba^*$.*

Solution : The regular expression

$$r = b + ba^* \text{ can be broken into } r_1 \text{ and } r_2 \text{ as}$$

$$r_1 = b$$

$$r_2 = ba^*$$

Let us draw the NFA for r_1, which is very simple.

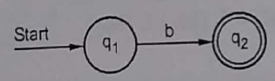


Fig. 2.4.5 For r_1

Now, we will go for $r_2 = ba^*$, this can be broken into r_3 and r_4 where $r_3 = b$ and $r_4 = a^*$. Now the case for concatenation will be applied. The NFA will look like this r_3 will be shown in Fig. 3.7.

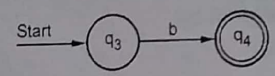


Fig. 2.4.6 or r_3

and r_4 will be shown as

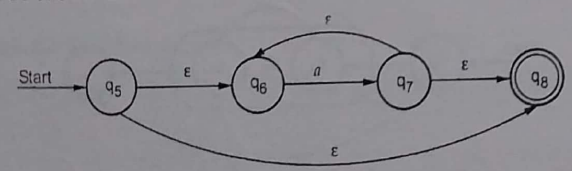


Fig. 2.4.7 For r_4

The r_2 will be $r_2 = r_3 \cdot r_4$

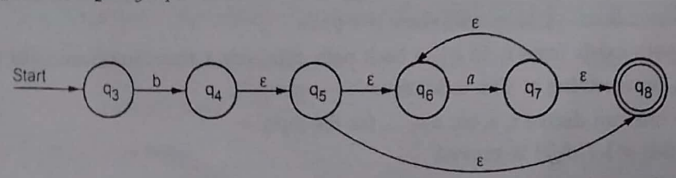


Fig. 2.4.8 For r_2

Now, we will draw NFA for $r = r_1 + r_2$ i.e. $b + ba^*$

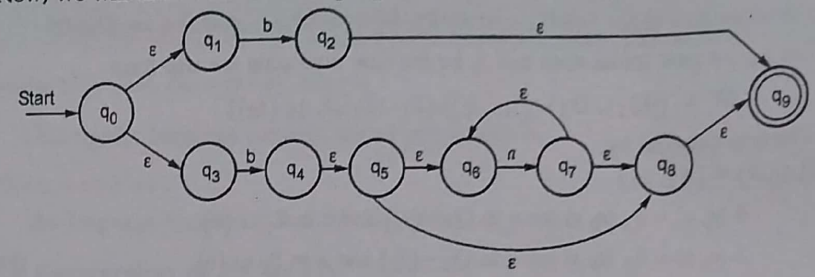


Fig. 2.4.9

Example 2.4.2 *Construct NFA with ε moves for the regular expression $(0+1)^*$.*

Solution : The NFA will be constructed step by step by breaking regular expression into small regular expressions.

$$r_3 = (r_1 + r_2)$$

$$r = r_3^*$$

where $r_1 = 0$, $r_2 = 1$

NFA for r_1 will be

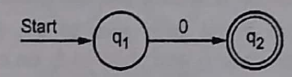


Fig. 2.4.10

NFA for r_2 will be

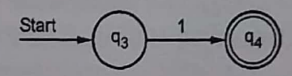


Fig. 2.4.11

NFA for r_3 will be

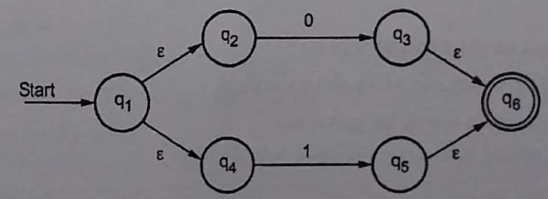


Fig. 2.4.12

Scanned with OKEN Scanner

And finally

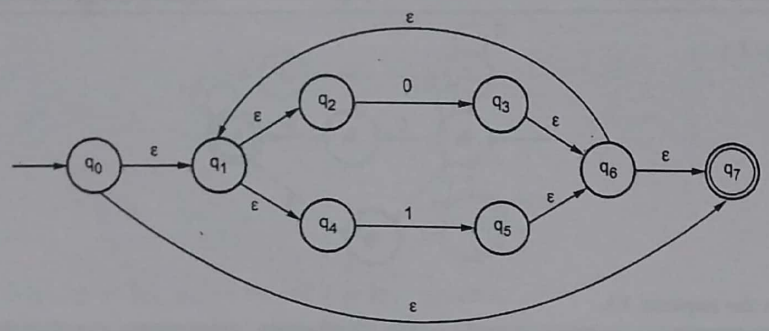


Fig. 2.4.13

2.4.1 Construction of FA from given RE using Direct Method

This method is a direct method for obtaining FA from given regular expression. This is called a **subset** method. The method is given as below -

Step 1 : Design a transition diagram for given regular expression, using NFA with ε moves.

Step 2 : Convert this NFA with ε to NFA without ε.

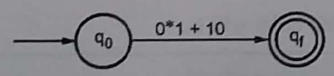
Step 3 : Convert the obtained NFA to equivalent DFA.

Let us understand this method with the help of some example.

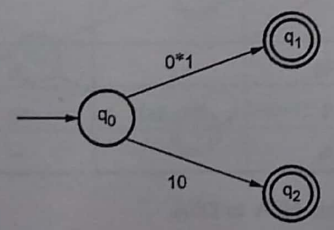
Example 2.4.3 *Construct FA for regular expression $0^*1 + 10$.*

Solution : We will construct FA r.e. = $0^*1 + 10$ as follows -

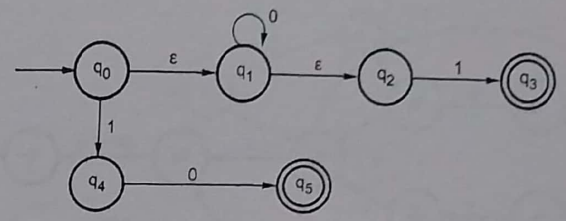
Step 1 :



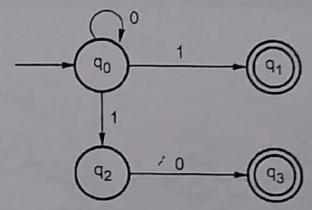
Step 2 :



Step 3 :



Step 4 :

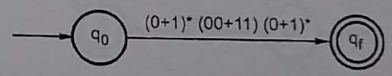


is the required FA.

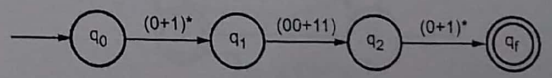
Example 2.4.4 *Construct finite automaton to accept the regular expression $(0 + 1)^*(00 + 11)(0 + 1)^*$.* **SPPU : Dec.-12, Marks 6**

Solution : The FA for r.e. = $(0 + 1)^*(00 + 11)(0 + 1)$ as follows -

Step 1 :

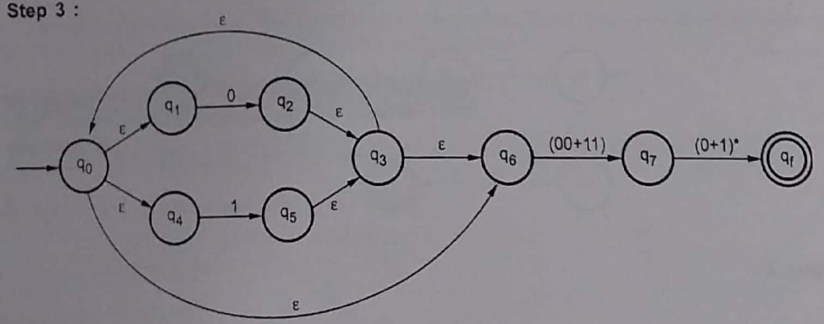


Step 2 :

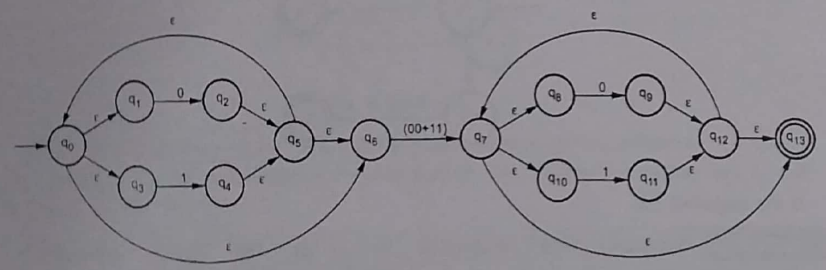


Scanned with OKEN Scanner

Step 3 :

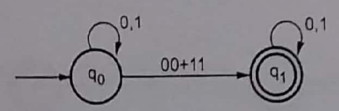


Step 4 :

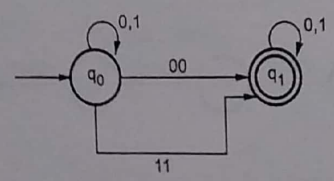


Step 5 :

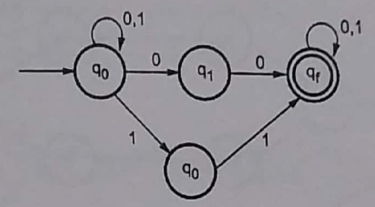
We will remove ε - transitions.



Step 6 :



Step 7 :

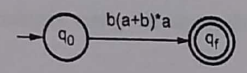


is the required FA.

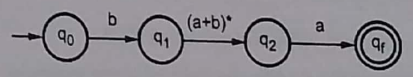
Example 2.4.5 *Construct DFA for following regular expression (RE)*
$RE = b(a+b)^*a$. **SPPU : Dec.-10, Marks 6**

Solution : We will construct DFA as follows -

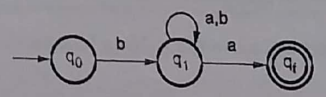
Step 1 :



Step 2 :



Step 3 :



The transition table for this NFA will be

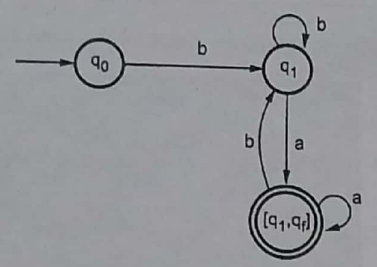
State \ Input	a	b
q_0	–	q_1
q_1	(q_1, q_f)	q_1
q_f	–	–

Step 4 : We will convert this NFA to DFA.

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

$\delta(q_0, b) = q_1$

Scanned with OKEN Scanner



$\delta(q_1, a) = \{q_1, q_f\} \rightarrow$ we call it as $[q_1, q_f]$ state.

$\delta(q_1, b) = q_1$

As new state $[q_1, q_f]$ is obtained we will obtain input transitions on it.

$\delta([q_1, q_f], a) = [q_1, q_f]$

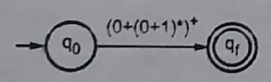
$\delta([q_1, q_f], b) = q_1$

As no new state is generated we can construct DFA with $[q_1, q_f]$ state. The $[q_1, q_f]$ is a final state. This is the required DFA.

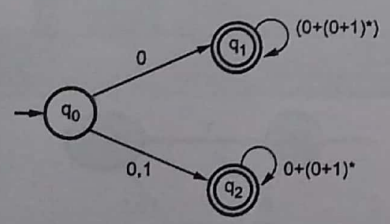
Example 2.4.6 *Construct NFA for the following regular expression.*
$(0+(0+1)^*)^+$ **SPPU : May-11, Marks 6**

Solution :

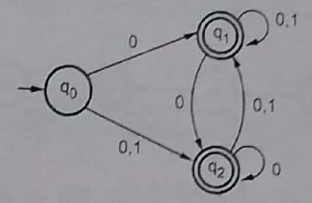
Step 1 :



Step 2 :



Step 3 :



This is the required NFA.

Example 2.4.7 *Construct NFA from the following regular expressions :*
i) $0^*1^*2^*$ *ii)* $(00+1)^*(10)^*$. **SPPU : Dec.-11, Marks 8; Oct.-18, Marks 5**

Solution :

i)

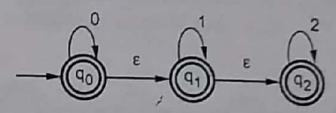


Fig. 2.4.14

ii) The NFA can be r1·r2

Step 1 :

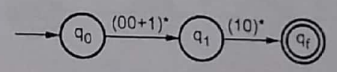


Fig. 2.4.14 (a)

Step 2 :

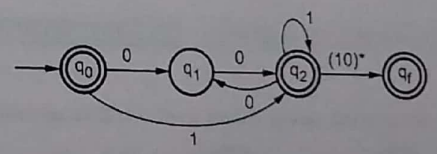


Fig. 2.4.14 (b)

Scanned with OKEN Scanner

Step 3 :

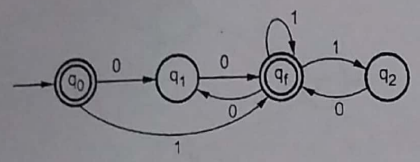


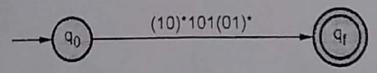
Fig. 2.4.14 (c)

Example 2.4.8 *Construct a FA for given regular expression (10)* 101 (01)**

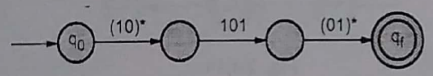
SPPU : May-16, End Sem, Marks 4

Solution :

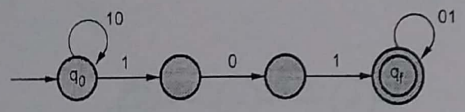
Step 1 : We will construct using direct method.



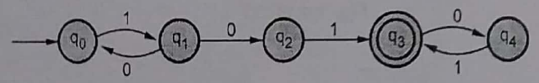
Step 2 :



Step 3 :



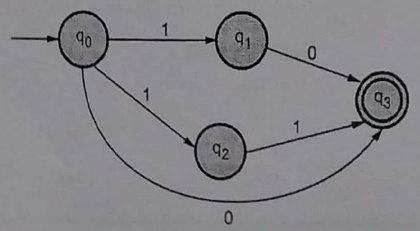
Step 4 :



Example 2.4.9 *Construct DFA for R.E. 10 + (0 + 11)* **SPPU : Dec.-16, End Sem, Marks 6**

Solution :

Step 1 : The NFA can be constructed using direct method. It is as follows :



Step 2 : The transition table for the above drawn NFA is as follows :

State \ Input	0	1
→ q_0	$[q_3]$	$[q_1, q_2]$
q_1	q_3	ϕ
q_2	ϕ	$[q_3]$
q_3	ϕ	ϕ

Step 3 : Now we will apply δ transition on each state for each input symbol.

$\delta[q_0, 0] = [q_3]$

$\delta[q_0, 1] = [q_1, q_2]$ ← **New state**

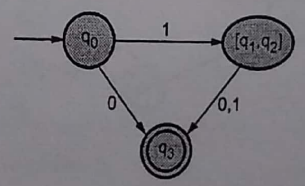
$\delta[q_1, q_2] = [q_3]$

$\delta[[q_1, q_2], 1] = [q_3]$

- It is observed from above transitions, that the only reachable states are $[q_3]$ and $[q_1, q_2]$. Hence we will now obtain δ transitions for these states.
- The state $[q_2]$ and $[q_1]$ seem to be unreachable states.
- The transition table will now be :

State \ Input	0	1
$[q_0]$	$[q_3]$	$[q_1, q_2]$
$[q_1, q_2]$	$[q_3]$	$[q_3]$
$[q_3]$	ϕ	ϕ

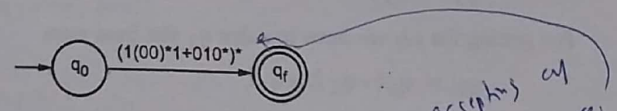
Step 4 : The DFA will be



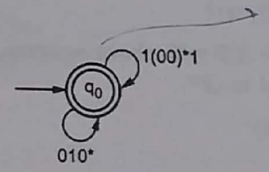
Scanned with OKEN Scanner

Example 2.4.10 *Construct DFA for following r.e.*
$r = (1\ (00)^*\ 1 + 010^*)^*$ using direct method **SPPU : Aug.-17, In Sem, Marks 6**

Solution : Step 1 :

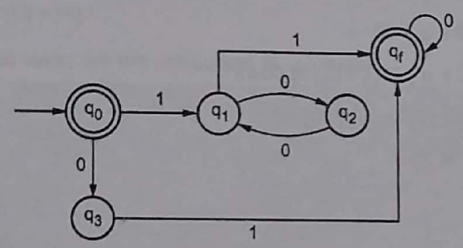


Step 2 :

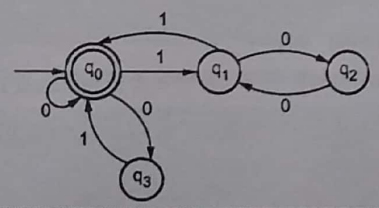


q_0 is accepting as we have closure

Step 3 :



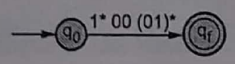
Step 4 :



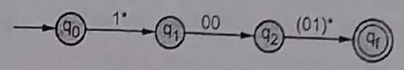
Example 2.4.11 *Construct the FA from given RE 1*00(01)**
SPPU : Dec.-19, End Sem, Marks 4

Solution :

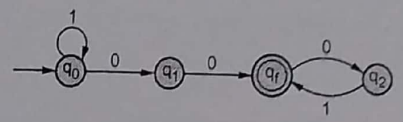
Step 1 :



Step 2 :



Step 3 :



2.5 Conversion of FA to RE using Arden's Theorem

SPPU : Dec.-10, May-11,14,15,19, Aug.-15,17, Oct.-16,18, Marks 8

Theorem 1 : Arden's Theorem : Let, P and Q be the two regular expressions over the input set Σ. The regular expression R is given as $R = Q + RP$.

Which has a unique solution as $R = QP^*$.

Proof : Let, P and Q are two regular expressions over the input string Σ.

If P does not contain ε then there exists R such that

$$R = Q + RP \qquad \text{... (1)}$$

We will replace R by QP^* in equation 1.

Consider R.H.S. of equation 1.

$$= Q + QP^*P = Q(\varepsilon + P^*P)$$

$$= QP^* \qquad \because \varepsilon + R^*R = R^*$$

Thus $\boxed{R = QP^*}$

is proved. To prove that $R = QP^*$ is a unique solution, we will now replace L.H.S. of equation 1 by $Q + RP$. Then it becomes

$$Q + RP$$

But again R can be replaced by $Q + RP$.

$$\therefore \quad Q + RP = Q + (Q + RP)P$$

$$= Q + QP + RP^2$$

Again replace R by $Q + RP$.

$$= Q + QP + (Q + RP)P^2$$

$$= Q + QP + QP^2 + RP^3$$

Scanned with OKEN Scanner

Thus if we go on replacing R by Q + RP then we get,

$$Q + RP = Q + QP + QP^2 + ... + QP^i + RP^{i+1}$$

$$= Q(\varepsilon + P + P^2 + ... P^i) + RP^{i+1}$$

From equation 1,

$$R = Q(\varepsilon + P + P^2 + ... + P^i) + RP^{i+1} \quad ... (2)$$

Where $i \geq 0$

Consider equation 2,

$$R = Q\underbrace{(\varepsilon + P + P^2 + ... + P^i}_{P^*} + RP^{i+1}$$

$$\therefore \quad R = QP^* + RP^{i+1}$$

Let w be a string of length i.

In RP^{i+1} has no string of less than i + 1 length. Hence w is not in set RP^{i+1}. Hence R and QP^* represent the same set. Hence it is proved that

$$R = Q + RP \text{ has a unique solution.}$$

$$R = QP^*.$$

Problems based on FA to RE

Example 2.5.1 *Find out the regular expression from given DFA.*

SPPU : Dec.-10, May-15, End Sem. Marks 4, May-11, Aug.-17, Oct.-18, In Sem, Marks 6, May-19, End Sem, Marks 5

Solution : Let us solve the DFA by writing the regular expression, for each state.

$$q_1 = q_1 0 + q_3 0 + \varepsilon \quad \because \text{ Initial state}$$

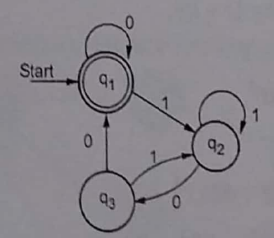


Fig. 2.5.1

$$q_2 = q_2 1 + q_3 1 + q_1 1$$

$$q_3 = q_2 0$$

For getting the r.e. we have to solve q_1 the final state.

$$q_2 = q_2 1 + q_2 01 + q_1 1$$

$$q_2 = q_2(1+01) + q_1 1$$

We will compare R = Q + R P with above equation, so $R = q_2$, $Q = q_1 1$, $P = (1+01)$ which ultimately gets reduced to QP*.

$$q_2 = q_1 1(1+01)^*$$

Substituting this value to q_1

$$q_1 = q_1 0 + q_3 0 + \varepsilon$$

$$= q_1 0 + q_2 00 + \varepsilon$$

$$= q_1 0 + q_1 (1(1+01)^*) 00 + \varepsilon$$

$$q_1 = q_1 (0 + 1(1+01)^* 00) + \varepsilon$$

Again $R = Q + RP$

where $R = q_1$

$Q = \varepsilon$

$P = 0 + 1(1+01)^* 00$

Hence $q_1 = \varepsilon \cdot [0 + 1(1+01)^* \cdot 00]^*$

$q_1 = [0 + 1(1+01)^* \cdot 00]^* \quad \because \varepsilon \cdot R = R$ is required r.e.

Example 2.5.2 *Construct r.e. for the DFA given in Fig. 2.5.2.* **SPPU : Dec.-10, Marks 4**

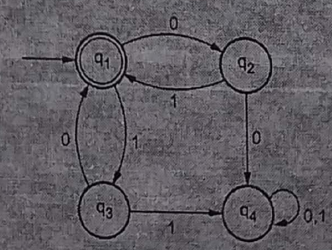


Fig. 2.5.2

Solution : Let us see the equations

$$q_1 = q_2 1 + q_3 0 + \varepsilon$$

Scanned with OKEN Scanner

$$q_2 = q_1 0$$

$$q_3 = q_1 1$$

$$q_4 = q_2 0 + q_3 1 + q_4 (0+1)$$

Let us solve q_1 first,

$$q_1 = q_2 1 + q_3 0 + \varepsilon$$

$$q_1 = q_1 01 + q_1 10 + \varepsilon$$

$$q_1 = q_1 (01+10) + \varepsilon \qquad \because R = Q + RP$$

$$q_1 = \varepsilon \cdot (01+10)^* \qquad \Rightarrow Q P^* \text{ where}$$

$$q_1 = (01+10)^* \qquad R = q_1,\ Q = \varepsilon,\ P = (01+10)$$

Thus the regular expression will be

$$r = (01+10)^*$$

Since q_1 is a final state, we are interested in q_1 only.

Example 2.5.3 *Construct regular expression for following transition diagram :*

SPPU : Dec.-10, Marks 6

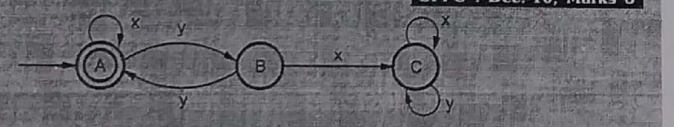


Fig. 2.5.3

Solution : We can get the regular expression from state A. We will write the equation of each state -

$$A = Ax + By + \varepsilon \qquad \text{... (1)}$$

$$B = Ay \qquad \text{... (2)}$$

$$C = Bx + Cx + Cy \qquad \text{... (3)}$$

The value of equation (2) is placed in equation (1).

$$A = Ax + Ayy + \varepsilon$$

$$\underset{R}{A} = \underset{R}{A}\,(\underbrace{x + yy}_{P}) + \underbrace{\varepsilon}_{Q}$$

$$A = \varepsilon\,(x+yy)^* \qquad \because R = Q + RP \Rightarrow R = QP^*$$

$$A = (x+yy)^*$$

Hence r.e. $= (x+yy)^*$

Example 2.5.4 *Construct RE by using Arden's Theorem for given DFA.*

SPPU : May-14, Marks 8

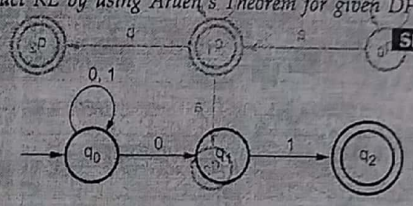


Fig. 2.5.4

Solution : We will write equations for each state

$$q_0 = q_0\,0 + q_0\,1 + \varepsilon \qquad \text{... (1)}$$

$$q_1 = q_0\,0 \qquad \text{... (2)}$$

$$q_2 = q_1\,1 \qquad \text{... (3)}$$

Consider equation (1)

$$q_0 = \underset{R}{q_0}\ \underset{R}{}\ \underbrace{(0+1)}_{P} + \underset{Q}{\varepsilon}$$

By Arden's Theorem, if R = Q+RP then R = QP*

$$q_0 = \varepsilon(0+1)^*$$

$$q_0 = (0+1)^* \qquad \text{... (4)}$$

$$q_1 = q_0 0$$

From equation (4)

$$q_1 = (0+1)^* 0 \qquad \text{... (5)}$$

$$q_2 = q_1\,1$$

Putting equation (5) in equation (3) we get

$$q_2 = (0+1)^* 01$$

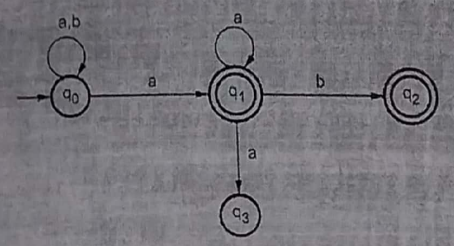
As q_2 is a final state, equation of q_2 becomes regular expression

$$\therefore \quad \text{r.e.} = (0+1)^*\,01$$

Scanned with OKEN Scanner

Example 2.5.5 *Construct a regular expression for given finite automata.*

SPPU : Aug.-15, Marks 6



Solution : We will write equations for each state

$$q_0 = q_0a + q_0b + \varepsilon \quad \text{... (1)}$$

$$q_1 = q_1a + q_0a \quad \text{... (2)}$$

$$q_2 = q_1b \quad \text{... (3)}$$

$$q_3 = q_3a \quad \text{... (4)}$$

As state q_1 and q_2 are two final states, hence these states will generate regular expression.

Solving equation (1)

$$\underset{R}{q_0} = \underset{R}{q_0}\,\underset{P}{(a+b)} + \underset{Q}{\varepsilon}$$

$$\therefore \quad q_0 = \varepsilon(a+b)^* = (a+b)^* \quad \text{... (5)}$$

Putting equation (5) in equation (2) we get -

$$\underset{R}{q_1} = \underset{R}{q_1}\,\underset{P}{q} + \underset{Q}{(a+b)^*a}$$

$$\therefore \quad q_1 = (a+b)^*\,aa^* \quad \text{... (6)}$$

Putting equation (6) in equation (3) we get

$$\therefore \quad q_2 = (a+b)^*\,aa^*b \quad \text{... (7)}$$

$$\therefore \quad \text{R. E.} = [(a+b)^*\,aa^*] + [(a+b)^*aa^*b]$$

Example 2.5.6 *Construct a regular expression corresponding to the following transition diagram using Arden's Theorem.*

SPPU : Oct. 16, In Sem, Marks 5

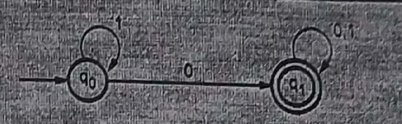


Fig. 2.5.5

Solution :

Step 1 : We will build the regular expression for each state.

$$q_0 = q_01 + \varepsilon \quad \text{... (1)}$$

$$q_1 = q_00 + q_1(0+1) \quad \text{... (2)}$$

Step 2 : The final state is q_1, hence the equation of q_1 state denotes the regular expression.

Hence we need to find out equation for state q_2

Step 3 : We will apply Arden's theorem to equation (1) First,

$$\underset{R}{q_0} = \underset{R}{q_0}\,\underset{P}{1} + \underset{Q}{\varepsilon}$$

As R = Q + RP gives R = QP*. We get

$$q_0 = \varepsilon 1^*$$

$$q_0 = 1^* \quad \text{...(3)} \because R^*E = R^*$$

$$\because \quad R^*\varepsilon = R^*$$

Step 4 : Using equation (3) for equation (2) we get

$$\underset{R}{q_1} = \underset{Q}{1^*0} + \underset{R}{q_1}\,\underset{P}{(0+1)}$$

Again by Arden's theorem R = QP*

$$\therefore \quad q_1 = 1^*0(0+1)^*$$

Step 5 : Thus we get final state equation i.e. equation of state q_1.

$$\therefore \quad \text{r.e.} = 1^*0(0+1)^*$$

Scanned with OKEN Scanner

2.6 Pumping Lemma for Regular Language

SPPU : May-09,11,16,18,19, Oct.-16,18, Dec.-08,12,14,16,19, Marks 6

- This is a basic and important theorem used for checking whether given string is accepted by regular expression or not. In short, this lemma tells us whether given language is regular or not.
- One key theme is that any language for which it is possible to design the finite automata is definitely the regular language.

Theorem 1 : Let L be a regular set. Then there is a constant n such that if z is any word in L and $|z| \geq n$ we can write z = u v w such that $|u\,v| \leq n$, $|v| \geq 1$ for all $i \geq 0$, $u\,v^i\,w$ is in L. The n should not be greater than the number of states.

Proof : If the language L is regular it is accepted by a DFA. $M = (Q, \Sigma, \delta, q_0, F)$. With some particular number of states say, n. Consider the input can be $a_1, a_2, a_3, \ldots a_m$, $m \geq n$. The mapping function δ could be written as $\delta(q_0, q_1, q_2, q_3, \ldots q_i) = q_i$.

The transition diagram is as shown in Fig. 2.6.1.

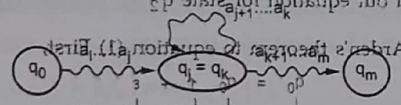


Fig. 2.6.1 Pumping lemma

If q_m is in F i.e. $q_1, q_2, q_3, \ldots q_m$ is in L(M) then $a_1, a_2, \ldots a_j\, a_{k+1}\, a_{k+2} \ldots\ldots a_m$ is also in L(M). Since there is path from q_0 to q_m that goes through q_j but not around the loop labelled $a_{j+1} \ldots a_k$. Thus

$$\delta(q_0, a_1, a_j\, a_{k+1} \ldots a_m) = \delta(\delta(q_0, q_1, \ldots q_j), a_{k+1} \ldots a_m)$$

$$= \delta(q_j, q_{k+1} \ldots q_m)$$

That is what we have proved i.e. given any long string can be accepted by FA, we should be able to find a substring near the beginning of the string that may be pumped i.e. repeated as many times as we like and resulting string may be accepted by FA.

The pumping lemma is used to check whether given language is regular or not.

2.6.1 Applications of Pumping Lemma

SPPU : May-09, 11, 16, Oct.-16, Dec.-08, 12, 14, 16, Marks 6

The pumping lemma is used to check whether given language is regular or not. We will discuss it with the help of some examples.

Example 2.6.1 *Show that the set $L = \{a^{i^2} \mid i > 1\}$ is not regular.*

SPPU : May-11, May-19, End Sem, Marks 6, Dec.-14, (End Sem.) 4 Marks

Solution : We have to prove that the language $L = a^{i^2}$ is not regular. This language is such that number of a's is always a perfect square.

For example, if we take i = 1

$$L = a^{1^2} = a \qquad \text{the length} = 1^2$$

$$= a^{2^2} = aaaa \qquad \text{length} = 2^2$$

and so on.

Now let us consider

$$L = a^{n^2} \quad \text{where} \quad \text{length} = n^2$$

it is denoted by z.

$$|z| = n^2$$

By pumping lemma z = u v w

where $1 \leq |v| \leq n$

As z = u v^i w where i = 1

Now we will pump v i.e. make i = z.

As we made i = 2 we have added one n.

$1 \leq |v| \leq n$.

$n^2 + 1 \leq |u\,v\,w| \leq n + n^2$

i.e. $n^2 + 1 \leq |u\,v\,w| \leq n^2 + n + n + 1$

i.e. $n^2 + 1 \leq |u\,v\,w| \leq (n+1)^2$

$= n^2 \leq |u\,v\,w| \leq (n+1)^2$

Thus the string lies between two consecutive perfect squares. But the string is not a perfect square. Hence we can say the given language is not regular.

For example,

$$L = a^{i^2}$$

Let i = 2

$$L = aaaa$$

Scanned with OKEN Scanner

$L = uvw$

Assume $uvw = aaaa$

Take $u = a$

$v = aa$

$w = a$

By pumping lemma, even if we pump v i.e. increase v then language should show the length as perfect square.

uvw

$= uv \cdot vw$

$= aaaaaa$

$=$ length of a is not a perfect square

Thus the behaviour of the language is not regular, as after pumping something onto it does not show the same property (being square for this example.)

Example 2.6.2 *Is the following language regular ? Justify your answer*
$L = \{0^{2n} \mid n \geq 1\}$.

Solution : This is a language length of string is always even.

i.e. $n = 1$; $L = 00$

$n = 2$ $L = 00\ 00$ and so on.

Let $L = uvw$

$L = 0^{2n}$

$|z| = 2n = u\,v^{i}\,w$

If we add 2n to this string length.

$|z| = 4n = uv \cdot vw$

$=$ even length of string.

Thus even after pumping 2n to the string we get the even length. So the language L is regular language.

Example 2.6.3 *Prove* $L = \{a^p \mid p \text{ is a prime}\}$ *is not regular.*

OR *State the pumping lemma theorem for given regular sets. Show that the language* $L = \{0^n \mid n \text{ is prime}\}$ *is not regular.* **SPPU : Dec-16, Oct-18, In Sem, Marks 4**

Solution : Let us assume L is a regular and p is a prime number.

$L = a^p$

$|z| = uvw \quad i = 1$

Now consider $L = u\,v^{i}\,w$ where $i = 2$

$= uv \cdot vw$

Adding 1 to p we get,

$p < |uvvw|$

$p < p+1$

But $p+1$ is not a prime number. Hence what we have assumed becomes contradictory. Thus L behaves as it is not a regular language.

Example 2.6.4 *Show that* $L = \{0^n 1^{n+1} \mid n > 0\}$ *is not regular.*
SPPU : May-18, End Sem, Marks 6

Solution : Let us assume that L is a regular language.

$|z| = |uvw|$

$= 0^n\ 1^{n+1}$

Length of string $|z| = n + n + 1 = 2n + 1$.

That means length is always odd.

By pumping lemma

$= |uv \cdot vw|$

That is if we add $2n + 1$

$2n + 1 < (2n + 1) + 2n + 1$

$2n + 1 < 4n + 2$

But if $n = 1$ then we obtain $4n + 2 = 6$ which is no way odd. Hence the language becomes irregular.

Even if we add 1 to the length of $|z|$, then

$|z| = 2n + 1 + 1$

$= 2n + 2$

$=$ even length of the string.

So this is not a regular language.

Scanned with OKEN Scanner

The simple way to know whether given language is regular or not is that try to draw finite automata for it, if you can easily draw the FA for the given L then that language is surely the regular otherwise not.

Example 2.6.5 *Show that $L = \{ww \mid w \in \{a, b\}\}$ is not regular.* **SPPU : Dec.-12, Marks 6**

Solution : Assume that the language L is regular. This also means that there exists some DFA with n states.

If we consider $w = a^n b$ i.e. string of {a, b}

then $ww = a^n b a^n b$ is in L

i.e. $|ww| = 2n+2 > n$

By pumping lemma, we can write

$ww = xyz$ with $|y| = 0$ $|xy| \le n$

iii) For checking regularity we need to find whether $xy^iz \in L$ or not.

Case 1 : $w = xy^iz$

we assume i = 0. This is a case with y having no b's.

Then $w = xy$ i.e. $y = a^k$

Y containing no b's at all

$w = a^m b a^n b$

where $m = n-k$

Then we can not write xz in the form of ww. Because w belongs to set {a, b}*.

Case 2 : This is a case with y having only one b.

Then assume

$w = xy^iz$

These w = xz with only one b.

But then $xz \in L$ as in language L there should be even number of a's and even number of b's. [we have assumed $|ww| = 2n + 2$].

Thus from case 1 and case 2 we get the contradictions. So L is not regular.

Example 2.6.6 *Prove whether the language is regular or not*

$0^m 1^n 0^{m+n} \mid m >= 1$ *and* $n >= 1$ **SPPU : May-09, 16, End Sem., Marks 6**

Solution : Assume L is regular and $w \in L$

Let $w = xyz$

Consider the string $w = 011000 \in L$

If we map w = xyz with above string then

$$w = \underbrace{0}_{x}\underbrace{11}_{y}\underbrace{000}_{z} = 0^1 1^2 0^3 \in L$$

By pumping lemma $xy^iz \in L$ for i > = 1.

If i = 2 then

$W = 0\ (11)\ (11)\ (000)$

$= 0^1 1^4 0^3 \notin L$ because $L = 0^m 1^n\ 0^{m+n}$

Thus our assumption of L being regular is wrong. Hence given language is not regular.

Example 2.6.7 *Find whether the language is regular or not*

$L = \{WW^R \in \{a, b\}\}$

Solution : Consider, $L = \{w \in \{a, b\} \mid w = w^R\}$

Consider $w = abab$

$w^R = baba$

Hence $L = ww^R$

The string z can be denoted by

$z = uvw$ where $|z| \ge n$ and $|uv| \le n$ $|v| \ge 1$.

We assume $z = \underbrace{ab}_{n}\underbrace{ab}_{n}\underbrace{ba}_{n}\underbrace{ba}_{n}$

We can split $z = uvw$ as

$$z = \underbrace{a}_{u}\underbrace{b}_{1}\underbrace{ab\,ba\,ba}_{w}$$

$|u| = n-1$

$|v| = 1$

i.e. $|uv| = |u| + |v|$

$= n - 1 + 1$

$= n$

Scanned with OKEN Scanner

Scanned with OKEN Scanner

Now according to pumping lemma,

when $z = uv^iw \in L$ then language is said to be regular. We will assume that $L = ww^R$ is regular. And we will find uv^iw.

$\therefore \quad z = uv^iw = ababbaba$

Assume $\quad i = 0$ i.e. there will be

$z = uv^0w = uw$

Then the string becomes

$z = aabbaba$

$\neq ww^R$

i.e. $\quad z = aabbaba \notin L \quad$ ...(1)

Now consider $i = 2$ then

$z = uv^2w$

$= abbabbaba$

$\neq ww^R$

i.e. $\quad z = abbabbaba \notin L \quad$ ...(2)

From equations (1) and (2) we can state that

$z = uv^iw \notin L$

Hence our assumption that $L = ww^R$ being regular is wrong. Hence we can prove that

$z = ww^R$ is not regular.

Example 2.6.8 *Show that $L = \{0^i1^i \mid i \geq 1\}$ is not regular, by using pumping lemma.*

SPPU : May-19, End Sem, Marks 6

Solution : Consider $L = \{0^i1^i \mid i \geq 1\}$.

Let $z = 0011$

The string z can also be denoted as

$z = uvw$ where $\quad |z| \geq n$

$|uv| \leq n \; |v| \geq 1$

We can split

$z = uvw$ as

$$z = \underbrace{0}_{u}\underbrace{0}_{v}\underbrace{11}_{w}$$

Now according to pumping lemma, when $z = uv^iw \in L$ then language is said to be regular. We assume

$w = 0^i1^i$ is regular.

Case 1 : $z = uv^iw = 0011$

Assume $i = 0$, then $z = uv^0w = uw = 011 \notin L$

Case 2 : $z = uv^iw = 0011$

Assume $i = 2$, then

$z = uvvw = 00011 = 0^31^2 \notin L$

From above two cases, clearly $L = 0^i1^i$ is not regular.

University Questions

1. *State and Prove Pumping Lemma Theorem ?*

SPPU : Dec.-08, Marks 6; Dec.-19, End Sem, Marks 4

2. *Define pumping lemma for regular languages* **SPPU : Oct.-16, In Sem, Marks 2**

2.7 Closure Properties of Regular Language

If certain languages are regular and language L is formed from them by certain operations (such as union or concatenation) then L is also regular. These properties are called closure properties of regular languages. Such languages represent the class of regular laguages which is closed under the certain specific operations.

The closure properties express the idea that when one or many languages are regular then certain related languages are also regular. The closure properties of regular languages are as given below.

1. The union of two regular languages is regular.
2. The intersection of two regular languages is regular.
3. The complement of a regular languages is regular.
4. The difference of two regular languages is regular.
5. The reversal of a regular languages is regular.
6. The closure operation on a regular language is regular.
7. The concatenation of regular language is regular.

Scanned with OKEN Scanner

8. A homomorphism of regular languages is regular.
9. The inverse homomorphism of regular language is regular.

Theorem 1 : If L_1 and L_2 are two languages then $L_1 \cup L_2$ is regular.

Proof : If L_1 and L_2 are regular then they have regular expression $L_1 = L(R_1)$ and $L_2 = L(R_2)$. Then $L_1 \cup L_2 = L(R_1 + R_2)$ thus we get $L_1 \cup L_2$ as regular language. (any language given by some regular expression is regular).

Theorem 2 : The complement of regular language is regular.

Proof :

- Consider L_1 be regular language which is accepted by a DFA $M = (Q, \Sigma, \delta, q_0, F)$.
- The complement of regular language is $\overline{L}_1$ which is accepted by $M' = (Q, \Sigma, \delta, q_0, Q - F)$.
- That means M is a DFA with final states $\in$ F and M' is a DFA in which all the non-final states of M become final.
- In other words, we can say that the strings that are accepted by M are rejected by M' similarly, the strings rejected by M are accepted by M'.
- Thus as $\overline{L}_1$ is accepted by DFA M', it is regular.

Theorem 3 : If L_1 and L_2 are two regular languages then $L_1 \cap L_2$ is regular.

Proof : Consider that languages L_1 is regular. That means there exists some DFA M_1 that accepts L_1. We can write $M = (Q_1, \Sigma, \delta_1, q_1, F_1)$.

Similarly being L_2 regular there is another DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$

- Let, L be the language obtained from $L_1 \cap L_2$. We can the simulate $M = (Q, \Sigma, \delta, q, F)$.

where $Q = Q_1 \times Q_2$

$\delta = \delta_1 \times \delta_2$ a mapping function derived from both the DFAS

$q \in Q$ which is initial state of machine M.

$F = F_1 \cap F_2$, the set of final states, which is common for M_1 and M_2 both.

- It is clear that there exists some DFA which accepts $L_1 \cap L_2$ i.e. L. Hence L is a regular language. This proves that if L_1 and L_2 are two regular languages then $L_1 \cap L_2$ is regular. In other words the regular language is closed under intersection.

Theorem 4 : If L_1 and L_2 are two regular languages then $L_1 - L_2$ is regular.

Proof : The $L_1 - L_2$ can also be denoted as $L_1 \cup L_2$

- Consider L_1 be regular language which is accepted by DFA $M = (Q, \Sigma, \delta, q_0, F)$.

- The complement of regular language L_1 is $\overline{L}_1$ which is accepted by $M' = (Q, \Sigma, \delta, q_0, Q - F)$. That means M is a DFA with final state's set F and M' is a DFA in which all the non final states of M become final states and all the final states of M become non-final states.
- Thus L_1 and $\overline{L}_2$ are two regular languages. That also means : these languages are accepted by regular expressions. If $L_1 = L(R_1)$ and $\overline{L}_2 = L(R'_2)$. Then $L_1 \cup \overline{L}_2 = L(R_1 + R'_2)$.
- This ultimately shows that $L_1 \cup \overline{L}_2$ is regular. In other words $L_1 - L_2$ is regular. Thus regular languages are closed under difference.

Theorem 5 : The reversal of a regular languages is regular.

Proof :

- Reversal of a string means obtaining a string which is written from backward that is w^R is denoted as reversal of string w.
- That means $L(w^R) = (L(w))^R$.
- This proof can be done with basis of induction.

Basis : If $w = \varepsilon$ or ϕ then w^R is also ε or ϕ

i.e. $(\varepsilon)^R = \varepsilon$ and $(\phi)^R = \phi$

Hence $L(w^R)$ is also regular.

Induction :

Case 1 : If $w = w_1 + w_2$ then

$$w = (w_1)^R + (w)^R$$

As the regular language is closed under union. Then w is also regular.

Case 2 : If $w = w_1 w_2$

Consider $w_1 = (ab, bb)$

and $w_2 = (bbba, aa)$

The $w_1^R = (ba, aa)$

$w_2^R = (aaab, bb)$ then

$$w = w_1^R w_2^R$$

(ba, aa, aaab, bb) is also regular. Thus given language is regular one.

Scanned with OKEN Scanner

Theorem 6 : The closure operation on a regular language is regular.

Proof : If language L_1 is regular then it can be expressed as $L_1 = L(R_1^*)$. Thus for a closure operation a language can be expressed as a language of regular expressions. Hence L_1 is said to be a regular language.

Theorem 7 : If L_1 and L_2 are two languages then $L_1 \cdot L_2$ is regular. In other words regular languages are closed under concatenation.

Proof : If L_1 and L_2 are regular then they can be expressed as $L_1 = L(R_1)$ and $L_2 = L(R_2)$. Then $L_1 \cdot L_2 = L(R_1 \cdot R_2)$ thus we get a regular language. Hence it is proved that regular languages are closed under concatenation.

Theorem 8 : A homomorphism of regular languages is regular.

Proof : The term homomorphism means substitution of string by some other symbols.

For instance the string " aabb" can be written as 0011 under homomorphism. Clearly here, a is replaced by 0 and b is replaced by 1. Let $\sum$ is the set of input alphabets and Γ be the set of substitution symbols then $\sum^* \rightarrow \Gamma^*$ is homomorphism. The definition of homomorphism can be extended as

Let, $w = a_1 a_2 \ldots a_n$

$h(w) = h(a_1) h(a_2) \ldots h(a_n)$

If L is a language that belongs to set $\sum$, then the **homomorphic image** of L can be defined as :

$$h(L) = \{ h(w) : w \in L \}$$

To prove that if L is regular h(L) is also regular consider following example -

Let, $\sum = \{a, b\}$ and w = abab

Let $h(a) = 00$

and $h(b) = 11$

Then we can write h(w) = h(a) h(b) h(a) h(b)

= 00110011

The homomorphism to language is applied by applying homomorphism on each string of language.

$\therefore$ If $L = ab^*b$ then,

$L = \{ab, abb, abbb, abbbb, \ldots\}$

Now $h(L) = \{0011, 001111, 00111111, 0011111111, \ldots\}$

$\therefore$ $h(L) = 00\,(11)^+$

As it can be represented by a regular expression, it is a regular language. Hence it is proved that if L is regular then h(L) is also regular. In other words, family of regular languages is closed under homomorphism.

Theorem 9 : The inverse homomorphism of regular language is regular.

Proof : Let $\sum^* \rightarrow \Gamma^*$ is homomorphism.

- The $\sum$ is the input set and Γ be the substitution symbols used by homomorphic function.
- Let, L be the regular language where $L \in \sum$, then h(L) be homomorphic language.
- The inverse homomorphic language can be represented as $h^{-1}(L)$
- Let,

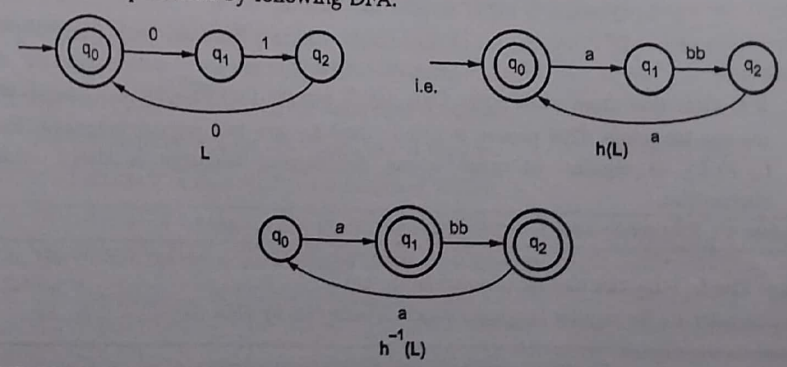
  $h^{-1}(L) = \{ w \mid w \in L \}$
- If L is regular then h(L) is also regular because regular language is closed under homomorphism. That if there exist a FA $M = (Q, \epsilon, \delta, q_0, F)$ which accepts L then h(L) must also be accepted by FA M. For complement of L i.e. language L' the inverse homomorphic language is $h^{-1}(L)$.
- Let M' be the FA in which all the final states of M become non-final states and all the non-final states of M become the final states.
- Clearly the language L' can be accepted by M' Hence $h^{-1}(L)$ must also be accepted by FA M'.

For example - Let L = (010)* be a regular language. And L = {010, 010010, ...}

Let h(0) = a and h(1) = bb be homomorphic function. Then

h(L) = (abba)*

This L can be represented by following DFA.



Scanned with OKEN Scanner

Thus there exists a FA which accepts $h^{-1}(L)$. This shows that, inverse homomorphism of regular language is regular.

2.8 Applications of Regular Expressions **SPPU : Oct.-16, Dec.-19, Marks 5**

The regular expressions can be modelled by finite automata. We have solved many examples and experienced that regular expressions are effective representation of languages. The smaller unit of regular expression can express the given language over certain input set.

There are following applications of regular expressions and finite automata.

1. Regular expressions in UNIX

- There are various UNIX notations used for regular expressions. These notations have many additional capabilities and features.
- These notations have ability to name and refer previous strings that have a matched pattern. This ability helps in recognizing nonregular languages.
- UNIX regular expressions allow to write character classes. There are some rules for these character classes which are as given below -

1) The symbol '.' stands for any single character.
2) The sequence [1, 2, 3, ... 10] means 1+2+3+...+10.
3) The range of characters can be specified using square brackets.

   For example : [a-z] denotes set of lower case letters.
4) For matching with some special character a backslash is used.

   For example \N means match with character N.
5) Special notations can be used to represent some common class of characters

   For example : [:digit:] represents the class [0-9]. The [:alpha:] is same as [A-Za-Z].
6) The operator | is used to denote union.
7) ? is used to denote preceding zero or single character.

   For example : - ? [0-9] means the number can be positive or negative number.
8) + is used to denote one or more occurrences.
9) * is used to denote zero or more occurrences.
10) {n} means n copies of.

   For example : a {3} means aaa.

2. Lexical analyzers : Compiler uses this program of lexical analyzer in the process of compilation. The task of lexical analyzer is to scan the input program and separate out the 'tokens'.

For example : Identifier is a category of token in the source language and it can be identified by regular expression as

(letter) (letter + digit)*

If anything in the source language matches with this regular expression then it is recognized as identifier. The letter is nothing bit a set {A, B, ... Z, a, b, ... z} * and digit is {0, 1, ... 9} *. The regular expression is effective way for identifying token from a language.

We can represent the above regular expression using a finite automata as follows-

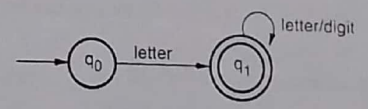


Fig. 2.8.1

3) Finding patterns in text

Regular expressions are useful notations used for finding the patterns from given text. A large class of pattern can be identified by regular expression.

For example : The strings ending with digits is a pattern in the given text which can be recognized by following regular expression -

[a-zA-Z] + [0-9] +

Thus use of regular expressions for identifying patterns from given text makes the job of compiler more simplified.

In web applications also the pattern matching is done using regular expressions.

Limitations of FA

The main limitation of FA is that it can not remember arbitrarily long input sequence because it does not contain any memory. Hence it can not solve following types of problems.

1. Checking well-formedness of parenthesis.
2. Checking the palindrome condition of given language.

University Question

1. *Write a note on applications of regular expressions.*

SPPU : Oct.-16, In Sem, Marks 5; Dec.-19, End Sem, Marks 3

Scanned with OKEN Scanner

2.9 Fill in the Blanks

Q.1 The zero or more occurrences of a letter is denoted using ___ operator.

Q.2 The one or more occurrences of a letter is denoted using ___ operator.

Q.3 The set of strings that end in 0 is denoted as ____.

Q.4 The theorem for checking non regularity of a language is ____.

Q.5 If P, Q, R are three regular expressions and if R = Q + RP then it has a unique solution given by ____.

Q.6 The RE in which any number of 0's is followed by any number of 1's followed by any number 2's is ____.

Q.7 In regular expression for adjoining two symbols we use ___ operator.

Q.8 In regular expression the null or empty string is denoted by ____.

Q.9 Concatenation of R with Ø gives ____.

Q.10 Ø* = ____.

Q.11 If A and B are regular languages then A* is ____.

2.10 Multiple Choice Questions

Q.1 Regular expressions are used to represent which language _____.

a recursive language
b context free language
c regular language
d all of these

Q.2 The set of all strings over Σ = {a,b} in which all strings of a's and b's ending in bb is _____.

a ab
b a*bbb
c (a+b)* bb
d all of these

Q.3 Which of the following is not a prefix of abc ?

a ε
b a
c ab
d bc

Q.4 What is R+R _____.

a R
b 0
c R+
d None

Q.5 Any transition graph has an equivalent _____.

a RE
b DFA
c NFA
d DFA, NFA, RE

Q.6 Which of the following is not a regular expression ?

a (a+b)*abb
b abba*
c 0 + 1
d (a+b)*-(aa)

Q.7 Regular expressions are _____.

a type 0 language
b type 1 language
c type 2 language
d type 3 language

Q.8 Regular expressions are closed under _____.

a union
b intersection
c kleen star
d all of these

Q.9 Regular expression(a/b)(a/b) denotes the set of _____.

a {ab,ab}
b {aa,bb,ab,ba}
c {aa,bb}
d {a,b}

Q.10 Which of the following identity is wrong ?

a R + R = R
b (R*)* = R*
c εR = Rε = R
d ØR = RØ = RR*

Q.11 Concatenation operation refers to which of the following set operations :

a Union
b Dot
c Kleene
d a and c

Answer Keys for Fill in the Blanks :

Q.1	*	Q.2	+
Q.3	(0 + 1)*0	Q.4	pumping lemma
Q.5	QP*	Q.6	0*1*2*
Q.7	dot or	Q.8	*

Scanned with OKEN Scanner