RGPVNOTES.IN

Program : **B.Tech**

Subject Name: **Theory of Computation**

Subject Code: **CS-501**

Semester: **5th**

Subject Notes
CS501- Theory of Computation

B. Tech, CSE-5th Semester

Unit -2

Syllabus: Types of Finite Automata: Non-Deterministic Finite Automata (NDFA), Deterministic finite automata machines, conversion of NDFA to DFA, minimization of automata machines, regular expression, Arden's theorem. Meaning of union, intersection, concatenation and closure, 2-way DFA.

Unit Objective: Relate practical problems to languages, automata, computability and complexity. Constructs abstract models of computing and check their power to recognize the language.

……

Finite Automata: An automaton with a finite number of states is called a Finite Automaton (FA) or FiniteState Machine (FSM).

An automaton has a mechanism to read input, which is string over a given alphabet. This input isactually written on an input tape /file, which can be read by automaton but cannot change it. Input file is divided into cells each of which can hold one symbol. Automaton has a control unitwhich is said to be in one of finite number of internal states.

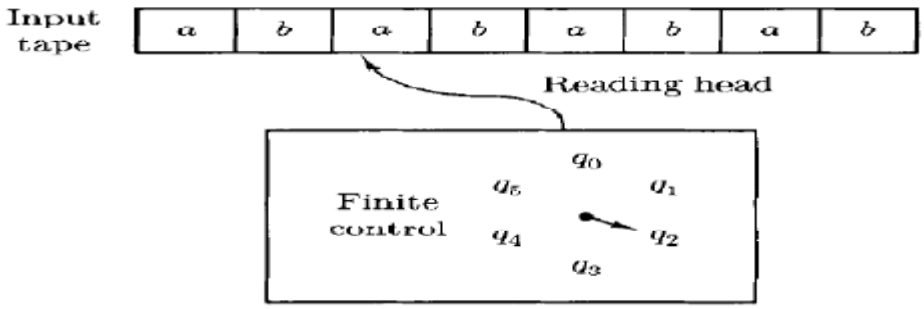


Figure 2.1: Finite Automata

Formal definition of Finite Automata:

A finite automaton is a 5-tuple M = (Q, Σ, δ, q, F), where

1. Q is a finite set, whose elements are called states,
2. Σ is a finite set, called the alphabet; the elements of Σ are called symbols,
3. δ: Q × Σ → Q is a function, called the transition function,
4. q is an element of Q; it is called the start state or initial state,
5. F is a subset of Q; the elements of F are called accept states or final state.

Related Terminologies:

Alphabet: An alphabet is any finite set of symbols.

Example: Σ = {a, b, c, d} is an alphabet set where 'a', 'b', 'c', and'd' are symbols.

String: A string is a finite sequence of symbols taken from Σ.

Example: 'cabcad' is a valid string on the alphabet set Σ = {a, b, c, d}

Length of a String: It is the number of symbols present in a string. (Denoted by |S|).

Examples: If S='cabcad', |S|= 6

If |S|= 0, it is called an empty string (Denoted by λ or ε)

Language: A language is a subset of Σ* for some alphabet Σ. It can be finite orinfinite.

Example: If the language takes all possible strings of length 2 over Σ = {a, b},then L = {ab, bb, ba, bb

1.1Types of Finite Automata:

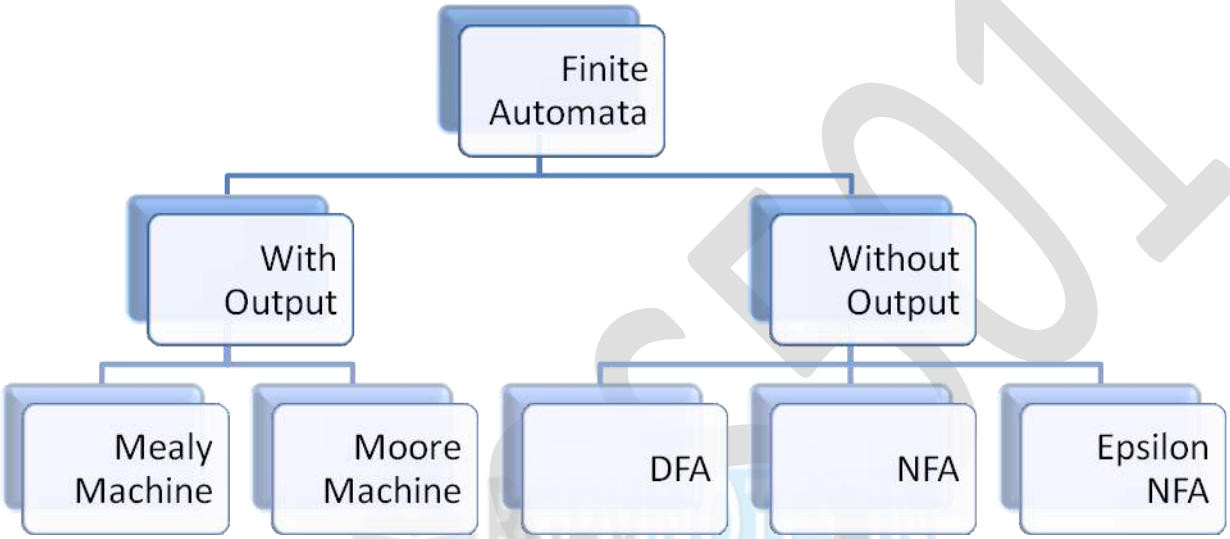


Figure 2.2: Classification of Finite Automata

An example of finite automata

Let A = {w: w is a binary string containing an odd number of 1s}.

We claim that this language A is regular. In order to prove this, we have to construct a finite automaton M such that A =L(M).

Steps to construct finite automata:

- The FA reads the input string w from left to right and keep scanning on the number of 1's
- After scanning the entire string, it counts the number of 1's to check whether it is odd or even
- If the number of 1's is odd then the string is acceptable otherwise it is rejected

Using this approach, the finite automaton needs a state for every integer i ≥ 0; indicating that the number of 1s read so far is equal to i. Hence, to design a finite automaton that follows this approach, we need an infinite number of states. But, the definition of finite automaton requires the number of states to be finite. A better, and correct approach, is to keep track of whether the number of 1s read so far is even or odd. This leads to the following finite automaton:

• The set of states is Q = $\{q_0, q_1\}$. If the finite automaton is in state q1, then it has an even number of 1's; if it is in state q0, then it has an odd number of 1's.

• The alphabet is Σ = {0, 1}.

• The start state is q0, because at the start, the number of 1's read by the automaton is equal to 0, and 0 is even.

• The set F of accept states is F = {q_1}.

• The **transition function δ** is given by the following table:

State	Input 0	Input 1
q_0	**q_0**	**q_1**
q_1	**q_1**	**q_0**

Table 2.1: Transition Table/Matrix

This finite automaton M = (Q, Σ, δ, q_0, F) can also be represented by its state diagram.

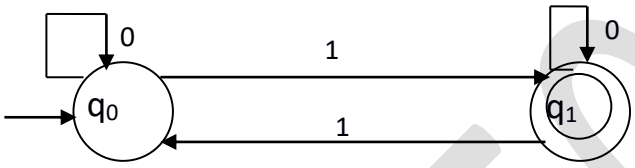


Figure2.3: Transition Graph

Transition Graph: it is a finite directed labeled graph in which each vertex (or node) represent a state and the directed edges indicate the transition of state. Edges are labeled with input symbol.

Transition Matrix: It is two-dimension matrixes between states of automata and Input symbol. Elements of matrix are state form mapping (Σ X Q) into Q.

Deterministic Finite Automata (DFA):

Deterministic automaton is one in which each move (transition from one state to another) is uniquelydetermined by the current configuration. If the internal states input and contents of the storage are known it is possible to predict the next (future) behavior of the automaton.

Formal Definition of a DFA

A DFA can be represented by **a 5-tuple (Q, Σ, δ, q0, F)** where:

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

DFA can be represented by Transition Graph and Transition Diagram as shown in Table 1.1 and Figure 1.4

Non-Deterministic Finite Automata (NDFA):

In NDFA, for a particular input symbol, the machine can move to any combination of thestates. In other words, the exact state to which the machine moves cannotbe determined. Hence, it is called Non-deterministic Automaton.

Formal Definition of NDFA

An NDFA can be represented by a **5-tuple (Q, Σ, δ, q0, F)** where:

1. Q is a finite set of states.
2. Σ is a finite set of symbols called the alphabets.
3. δ is the transition function where δ: Q × Σ → 2Q

(Here the power set of Q's (2Q) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states)

4. q0 is the initial state from where any input is processed (q0 ∈ Q).
5. F is a set of final state/states of Q (F ⊆ Q).

Difference between DFA & NDFA:

S. No.	DFA	NFA
1.	DFA stands for deterministic finite automata.	NDFA stands for non-deterministic finite automata.
2.	when processing a string in DFA , there is always a unique state to go next when each character is read. It is because for each state in DFA , there is exactly one state that corresponds to each character being read.	In NDFA several choices may exist for the next state. Can move to more than one states.
3.	DFA can not use empty string transition.	NDFA can use empty string transition.
4.	In DFA we cannot move from one state to another without consuming a symbol.	NDFA allows € (null) as the second argument of the transition function. This means that the NDFA can make a transition without consuming an input symbol.
5	For every symbol of the alphabet, there is only one state transition in DFA.	We do not need to specify how does the NDFA react according to some symbol.
6	DFA can understood as one machine.	NDFA can be understood as multiple title machines computing at the same time.
7	DFA will reject the string if it end at other than accepting state	If all the branches of NDFA dies or rejects the string, we can say that NDFA reject the string.
8	It is more difficult to construct DFA.	NDFA is easier to construct.
9	DFA requires more space.	NDFA requires less space.
10	For every input and output we can construct DFA machine.	It is not possible to construct an NDFA machine for every input and output.

Figure 2.4: Difference between DFA & NDFA

Comparison between Deterministic Finite Automata (DFA) and the Nondeterministic Finite Automata (NFA):

S. No.	Title	NFA	DFA

1.	Power	Same	Same
2.	Supremacy	Not all NFA are DFA.	All DFA are NFA
3.	Transition Function	Maps $Q \rightarrow (\sum \cup \{\lambda\} \rightarrow 2^Q)$, the number of next states is zero or one or more.	$Q \times \sum \rightarrow Q$, the number of next states is exactly one
4.	Time complexity	The time needed for executing an input string is more as compare to DFA.	The time needed for executing an input string is less as compare to NFA.
5.	Space	Less space requires.	More space requires.

Table No. 2.2 Comparison between NFA and DFA

Equivalence of DFA and NDFA:

Although the DFA and NFA have distinct definitions, an NFA can be translated to equivalent DFA using the subset construction algorithm. i.e., the constructed DFA and the NFA recognize the same formal language. Both types of automata recognize only regular languages.

Therefore, every language that can be described by some NDFA can also be described by some DFA.

Theorem

Let M = (Q, Σ, δ, q_0, F) be a non-deterministic finite automaton. There exists a deterministic finite automaton M', such that L(M') =L(M).

i.e. for every NDFA there exists a DFA which simulates the behavior of NDFA. Hence if language L is accepted by N9DFA, then there exist a DFA M' which also accept L. Where M' = (Q', Σ, δ', q'_0, F')

Conversion from NFA to DFA:

In NFA, when a specific input is given to the current state, the machine goes to multiple states. It can have zero, one or more than one move on a given input symbol. On the other hand, in DFA, when a specific input is given to the current state, the machine goes to only one state. DFA has only one move on a given input symbol.

Let, M = (Q, ∑, δ, q0, F) is an NFA which accepts the language L(M). There should be equivalent DFA denoted by M' = (Q', ∑', q0', δ', F') such that L(M) = L(M').

Steps for converting NFA to DFA:

Step 1: Initially Q' = φ

Step 2: Add q0 of NFA to Q'. Then find the transitions from this start state.

Step 3: In Q', find the possible set of states for each input symbol. If this set of states is not in Q', then add it to Q'.

Step 4: In DFA, the final state will be all the states which contain F (final states of NFA)

Example 1:

Convert the given NFA to DFA.

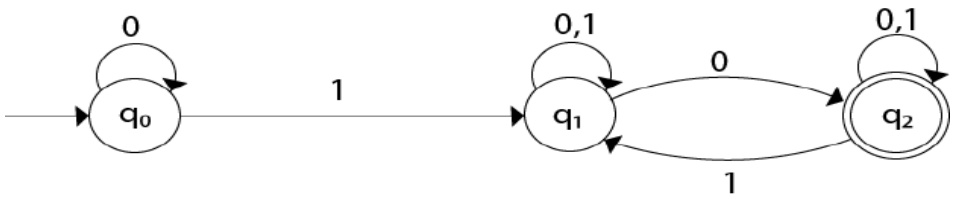


Figure No. 2.5 Transition Graph

Solution: For the given transition diagram we will first construct the transition table.

State	0	1
→q0	q0	q1
q1	{q1, q2}	q1
*q2	q2	{q1, q2}

Table No. 2.3 Transition Table

Now we will obtain δ' transition for state q0.

1. δ'([q0], 0) = [q0]
2. δ'([q0], 1) = [q1]

The δ' transition for state q1 is obtained as:

1. δ'([q1], 0) = [q1, q2] (new state generated)
2. δ'([q1], 1) = [q1]

The δ' transition for state q2 is obtained as:

1. δ'([q2], 0) = [q2]
2. δ'([q2], 1) = [q1, q2]

Now we will obtain δ' transition on [q1, q2].

1. δ'([q1, q2], 0) = δ (q1, 0) ∪ δ (q2, 0)
2. = {q1, q2} ∪ {q2}
3. = [q1, q2]
4. δ'([q1, q2], 1) = δ (q1, 1) ∪ δ (q2, 1)
5. = {q1} ∪ {q1, q2}
6. = {q1, q2}
7. = [q1, q2]

The state [q1, q2] is the final state as well because it contains a final state q2. The transition table for the constructed DFA will be:

State	0	1
→[q0]	[q0]	[q1]
[q1]	[q1, q2]	[q1]
*[q2]	[q2]	[q1, q2]

*[q1, q2]	[q1, q2]	[q1, q2]

Table No. 2.4 Transition Table

The Transition diagram will be:

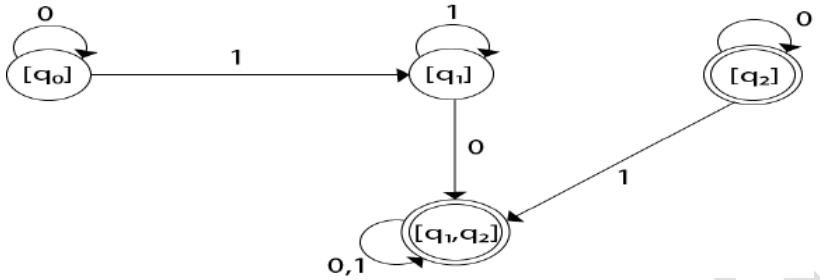


Figure No. 2.6 Transition Graph

The state q2 can be eliminated because q2 is an unreachable state.

Minimization of DFA

Minimization of DFA means reducing the number of states from given FA. Thus, we get the FSM(finite state machine) with redundant states after minimizing the FSM.

We have to follow the various steps to minimize the DFA. These are as follows:

Step 1: Remove all the states that are unreachable from the initial state via any set of the transition of DFA.

Step 2: Draw the transition table for all pair of states.

Step 3: Now split the transition table into two tables T1 and T2. T1 contains all final states, and T2 contains non-final states.

Step 4: Find similar rows from T1 such that:

1. 1. δ (q, a) = p
2. 2. δ (r, a) = p

That means, find the two states which have the same value of a and b and remove one of them.

Step 5: Repeat step 3 until we find no similar rows available in the transition table T1.

Step 6: Repeat step 3 and step 4 for table T2 also.

Step 7: Now combine the reduced T1 and T2 tables. The combined transition table is the transition table of minimized DFA.

Example:

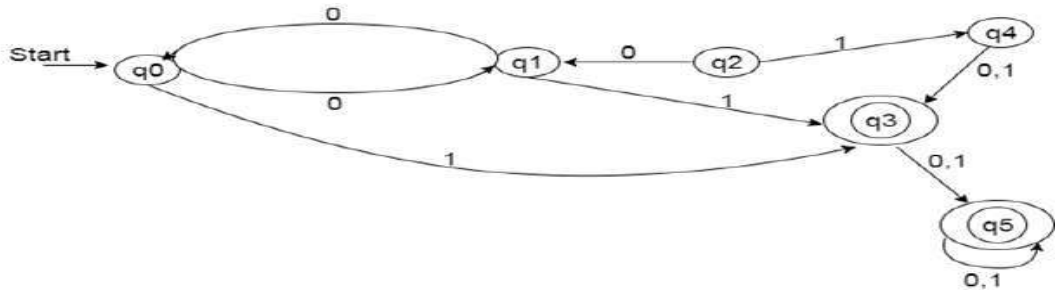


Figure 2.7: Transition Graph

Solution: Step 1: In the given DFA, q2 and q4 are the unreachable states so remove them.

Step 2: Draw the transition table for the rest of the states.

State	0	1
→q0	q1	q3
q1	q0	q3
*q3	q5	q5
*q5	q5	q5

Table 2.5: Transition Table/Matrix

Step 3: Now divide rows of transition table into two sets as:

1. One set contains those rows, which start from non-final states:

State	0	1
q0	q1	q3
q1	q0	q3

Table 2.6: Transition Table/Matrix

2. Another set contains those rows, which starts from final states.

State	0	1
q3	q5	q5
q5	q5	q5

Table 2.7: Transition Table/Matrix

Step 4: Set 1 has no similar rows so set 1 will be the same.

Step 5: In set 2, row 1 and row 2 are similar since q3 and q5 transit to the same state on 0 and 1. So skip q5 and then replace q5 by q3 in the rest.

State	0	1
q3	q3	q3

Table 2.8: Transition Table/Matrix

Step 6: Now combine set 1 and set 2 as:

State	0	1
→q0	q1	q3
q1	q0	q3
***q3**	q3	q3

Table 2.9: Transition Table/Matrix

Now it is the transition table of minimized DFA.

Regular Expression

- The language accepted by finite automata can be easily described by simple expressions called Regular Expressions. It is the most effective way to represent any language.
- The languages accepted by some regular expression are referred to as Regular languages.
- A regular expression can also be described as a sequence of pattern that defines a string.
- Regular expressions are used to match character combinations in strings. String searching algorithm used this pattern to find the operations on a string.

For instance:

In a regular expression, x* means zero or more occurrence of x. It can generate {e, x, xx, xxx, xxxx, ...}

In a regular expression, x^+ means one or more occurrence of x. It can generate {x, xx, xxx, xxxx, ...}

Operations on Regular Language

Properties of Regular Sets

Property 1. The union of two regular set is regular.

Proof –

Let us take two regular expressions

$RE_1 = a(aa)^*$ and $RE_2 = (aa)^*$

So, L_1 = {a, aaa, aaaaa,.....} (Strings of odd length excluding Null)

and L_2 ={ ε, aa, aaaa, aaaaaa,.....} (Strings of even length including Null)

$L_1 \cup L_2$ = { ε, a, aa, aaa, aaaa, aaaaa, aaaaaa,.....}

(Strings of all possible lengths including Null)

RE ($L_1 \cup L_2$) = a* (which is a regular expression itself)

Property 2. The intersection of two regular set is regular.

Proof –

Let us take two regular expressions

RE_1 = a(a*) and RE_2 = (aa)*

So, L_1 = { a,aa, aaa, aaaa, ...} (Strings of all possible lengths excluding Null)

L_2 = { ε, aa, aaaa, aaaaaa,.....} (Strings of even length including Null)

$L_1 \cap L_2$ = { aa, aaaa, aaaaaa,.....} (Strings of even length excluding Null)

RE ($L_1 \cap L_2$) = aa(aa)* which is a regular expression itself.

Property 3. The complement of a regular set is regular.

Proof –

Let us take a regular expression –

RE = (aa)*

So, L = {ε, aa, aaaa, aaaaaa, ...} (Strings of even length including Null)

Complement of **L** is all the strings that is not in **L**.

So, L' = {a, aaa, aaaaa, ...} (Strings of odd length excluding Null)

RE (L') = a(aa)* which is a regular expression itself.

Property 4. The difference of two regular set is regular.

Proof –

Let us take two regular expressions –

RE_1 = a (a*) and RE_2 = (aa)*

So, L_1 = {a, aa, aaa, aaaa, ...} (Strings of all possible lengths excluding Null)

L_2 = { ε, aa, aaaa, aaaaaa,.....} (Strings of even length including Null)

$L_1 - L_2$ = {a, aaa, aaaaa, aaaaaaa, ...}

(Strings of all odd lengths excluding Null)

RE ($L_1 - L_2$) = a (aa)* which is a regular expression.

Property 5. The reversal of a regular set is regular.

Proof –

We have to prove $\mathbf{L^R}$ is also regular if **L** is a regular set.

Let, L = {01, 10, 11, 10}

RE (L) = 01 + 10 + 11 + 10

L^R = {10, 01, 11, 01}

RE (L^R) = 01 + 10 + 11 + 10 which is regular

Property 6. The closure of a regular set is regular.

Proof –

If L = {a, aaa, aaaaa, ...} (Strings of odd length excluding Null)

i.e., RE (L) = a (aa)*

L* = {a, aa, aaa, aaaa , aaaaa,……} (Strings of all lengths excluding Null)

RE (L*) = a (a)*

Property 7. The concatenation of two regular sets is regular.

Proof –

Let RE_1 = (0+1)*0 and RE_2 = 01(0+1)*

Here, L_1 = {0, 00, 10, 000, 010, ...} (Set of strings ending in 0)

and L_2 = {01, 010,011,.....} (Set of strings beginning with 01)

Then, L_1 L_2 = {001,0010,0011,0001,00010,00011,1001,10010,.....}

Set of strings containing 001 as a substring which can be represented by an RE – (0 + 1)*001(0 + 1)*

Identities Related to Regular Expressions

Given R, P, L, Q as regular expressions, the following identities hold –

- Ø* = ε
- ε* = ε
- RR* = R*R
- R*R* = R*
- (R*)* = R*
- RR* = R*R
- (PQ)*P =P(QP)*
- (a+b)* = (a*b*)* = (a*+b*)* = (a+b*)* = a*(ba*)*

- R + Ø = Ø + R = R (The identity for union)
- R ε = ε R = R (The identity for concatenation)
- Ø L = L Ø = Ø (The annihilator for concatenation)
- R + R = R (Idempotent law)
- L (M + N) = LM + LN (Left distributive law)
- (M + N) L = ML + NL (Right distributive law)
- ε + RR* = ε + R*R = R*

Note: Two regular expressions are equivalent if languages generated by them are same. For example, (a+b*) * and (a+b) * generate same language. Every string which is generated by (a+b*) * is also generated by (a+b)* and vice versa.

Example 1:

Write the regular expression for the language accepting all combinations of a's, over the set ∑ = {a}

Solution:

All combinations of a's mean a may be zero, single, double and so on. If a is appearing zero times, that means a null string. That is, we expect the set of {ε, a, aa, aaa, ...}. So, we give a regular expression for this as:

1. R = a*

That is Kleene closure of a.

Example 2:

Write the regular expression for the language accepting all combinations of as except the null string, over the set ∑ = {a}

Solution:

The regular expression has to be built for the language

1. L = {a, aa, aaa,.....}

This set indicates that there is no null string. So, we can denote regular expression as:

$R = a^+$

Example 3:

Write the regular expression for the language accepting all the string containing any number of a's and b's.

Solution:

The regular expression will be:

1. r.e. = (a + b) *

This will give the set as L = {ε, a, aa, b, bb, ab, ba, aba, bab ...}, any combination of a and b.

The (a + b) * shows any combination with a and b even a null string.

Conversion of RE to FA

To convert the RE to FA, we are going to use a method called the subset method. This method is used to obtain FA from the given regular expression. This method is given 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.

Example 1:

Design a FA from given regular expression 10 + (0 + 11)0* 1.

Solution: First we will construct the transition diagram for a given regular expression.

Step 1:

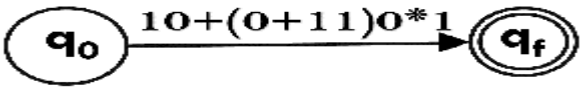


Figure 2.8: Transition Graph

Step 2:

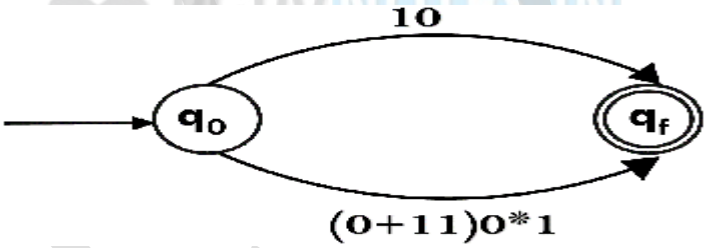


Figure 2.9: Transition Graph

Step 3:

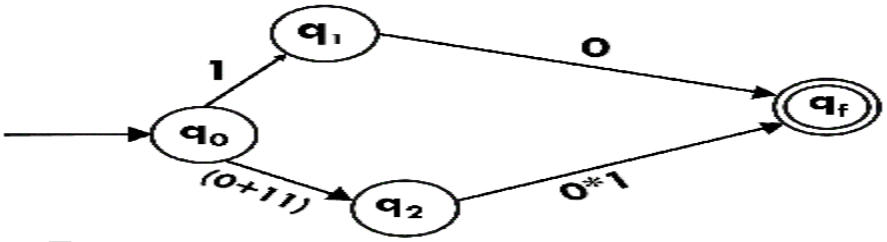


Figure 2.10: Transition Graph

Step 4:

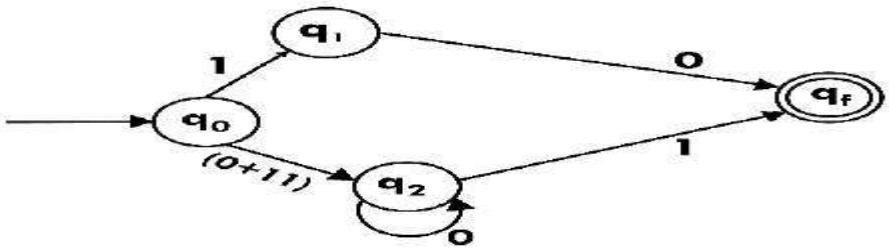


Figure 2.11: Transition Graph

Step 5:

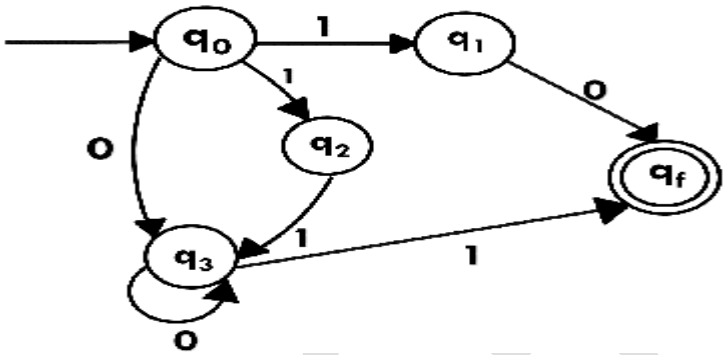


Figure 2.12: Transition Graph

Now we have got NFA without ε. Now we will convert it into required DFA for that, we will first write a transition table for this NFA.

State	0	1
→q0	q3	{q1, q2}
q1	qf	φ
q2	φ	q3
q3	q3	qf
*qf	φ	φ

Table 2.9: Transition Table/Matrix

The equivalent DFA will be:

State	0	1
→[q0]	[q3]	[q1, q2]
[q1]	[qf]	φ
[q2]	φ	[q3]
[q3]	[q3]	[qf]
[q1, q2]	[qf]	[qf]
*[qf]	φ	φ

Table 2.10: Transition Table/Matrix

Construction of an FA from a RE

Method

Step 1 Construct an NFA with Null moves from the given regular expression.

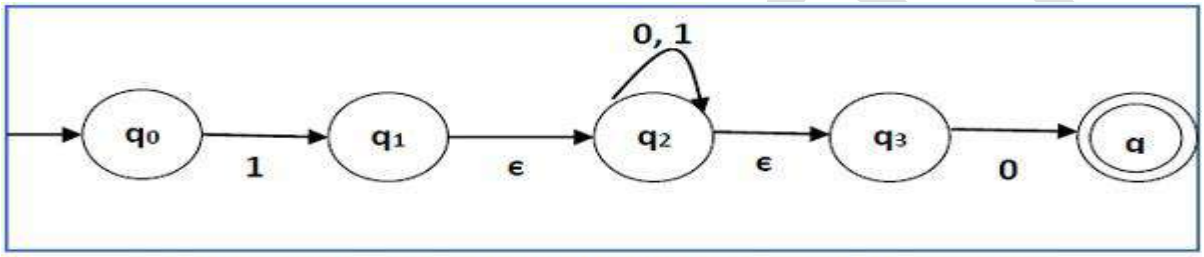
Step 2 Remove Null transition from the NFA and convert it into its equivalent DFA.

Problem

Convert the following RA into its equivalent DFA − 1 (0 + 1) * 0

Solution

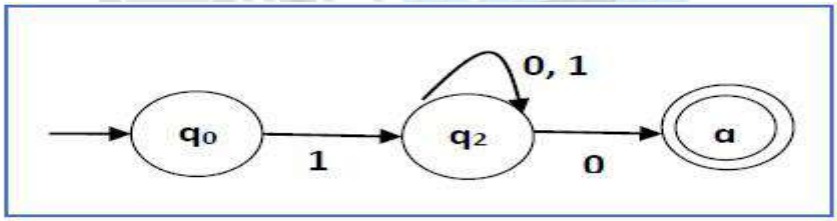
We will concatenate three expressions "1", "(0 + 1) *" and "0"



NDFA with NULL transition for RA: 1 (0 + 1)* 0

Figure 2.13: Transition Graph

Now we will remove the **ε** transitions. After we remove the **ε** transitions from the NDFA, we get the following −



NDFA without NULL transition for RA: 1 (0 + 1)* 0

Figure 2.14: Transition Graph

It is an NDFA corresponding to the RE − 1 (0 + 1) * 0. If you want to convert it into a DFA, simply apply the method of converting NDFA to DFA

Applications:

- Regular expressions are useful in a wide variety of text processing tasks, and more generally string processing, where the data need not be textual. Common applications include data validation, data scraping (especially web scraping), data wrangling, simple parsing, the production of syntax highlighting systems, and many other tasks.
- While regexps would be useful on Internet search engines, processing them across the entire database could consume excessive computer resources depending on the complexity and design of the regex.

Arden's Theorem:
In order to find out a regular expression of a Finite Automaton, we use Arden's Theorem along with the properties of regular expressions.

Statement:

Let P and Q be two regular expressions.
If P does not contain null string, then R = Q + RP has a unique solution that is R = QP*

Proof:
R = Q + (Q + RP) P [After putting the value R = Q + RP]
R= Q + QP + RPP
When we put the value of R recursively again and again, we get the following equation:
R = Q + QP + QP2 + QP3….
R = Q (ϵ + P + P2 + P3 + ….)
R = QP* [As P* represents (ϵ + P + P2 + P3 + ….)]
Hence, proved.

Assumptions for Applying Arden's Theorem:
1. The transition diagram must not have NULL transitions
2. It must have only one initial state

Following algorithm is used to build the regular expression form given DFA.
1. Let q_1 be the initial state.
2. There are q_2, q_3, q_4 .... qn number of states. The final state may be some q_j where j<= n.
3. Let α_{ji} represents the transition from q_j to q_i.
4. Calculate q_i such that
$q_i = \alpha_{ji} * q_j$
If q_j is a start state then we have:
$q_i = \alpha_{ji} * q_j + \varepsilon$
5. Similarly, compute the final state which ultimately gives the regular expression 'r'.

Example:
Construct the regular expression for the given DFA

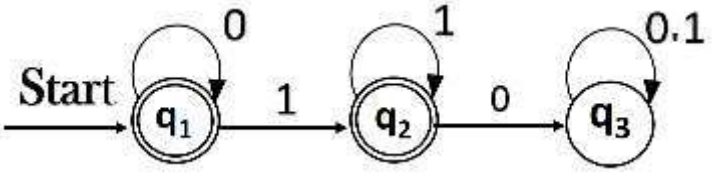


Figure 2.15: Transition Graph

Solution:
Let us write down the equations
q1 = q1 0 + ε
Since q1 is the start state, so ε will be added, and the input 0 is coming to q1 from q1 hence we write
State = source state of input × input coming to it
Similarly,
q2 = q1 1 + q2 1
q3 = q2 0 + q3 (0+1)
Since the final states are q1 and q2, we are interested in solving q1 and q2 only. Let us see q1 first
q1 = q1 0 + ε
We can re-write it as
q1 = ε + q1 0
Which is similar to R = Q + RP, and gets reduced to R = OP*.
Assuming R = q1, Q = ε, P = 0
We get

q1 = ε. (0) *
q1 = 0* (ε.R*= R*)
Substituting the value into q2, we will get
q2 = 0* 1 + q2 1
q2 = 0* 1 (1) * (R = Q + RP → Q P*)

The regular expression is given by
r = q1 + q2
= $0^* + 0^* 1.1^*$
r = $0^* + 0^* 1^+ (1.1^* = 1^+)$

Two-way finite automata

The finite automata discussed so far has a δ transition which indicate where to next from current state on receiving particular input. But 2-way FA is a model in which linear direction is mentioned on receiving particular and being in some current state. There are two direction that are allowed in 2-FA and these are left and right direction.

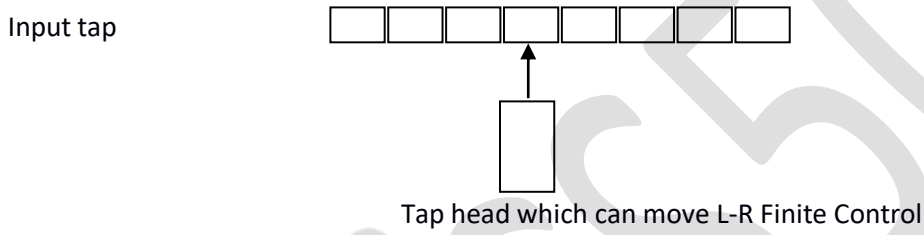


Figure 2.16: Two-way finite automata

Formal Definition of Two-way finite automata
It is a collection of 5-tuple M = (Q, Σ, δ, q_0, F), where
Q: a finite set of states
Σ: a finite set called the input alphabet
δ: a transition function which maps Q x {L,R}
q_0: a start state (also called initial state) which is an element of Q ($q_0 \in Q$)
F: Set of final states.

Example :

Consider the transition table .check the string "101001" whether it is accepted by Two - Way DFA or not?

States	0	1
-> q0	(q0, R)	(q1, R)
q1 (Final State)	(q1, R)	(q2, L)
q2	(q0, R)	(q2, L)

Table 2.11: Transition Table/Matrix

Acceptability of the string using two - way DFA is –

$$
\begin{aligned}
q_0101001 &\vdash 1q_101001 \\
&\vdash 10q_11001 \\
&\vdash 1q_201001 \\
&\vdash 10q_01001 \\
&\vdash 101q_1001 \\
&\vdash 1010q_101 \\
&\vdash 10100q_11 \\
&\vdash 1010q_201 \\
&\vdash 10100q_01 \\
&\vdash 101001q_1
\end{aligned}
$$

RGPVNOTES.IN

We hope you find these notes useful.

You can get previous year question papers at https://qp.rgpvnotes.in .

If you have any queries or you want to submit your study notes please write us at rgpvnotes.in@gmail.com

LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in