FINITE AUTOMATA & REGULAR EXPRESSIONS

3.1 Objectives

After studying this chapter, students will be able to

- Define Regular Grammar and Regular Expressions
- Construct various types of Regular Expressions
- Construct Finite Automata from Regular Expressions
- Construct Regular Expressions from Finite Automata
- Know ARDEN's theorem and its importance
- Construct Regular Grammar from Finite Automata
- Construct Finite Automata from Regular Grammar
- Know various Properties of Regular Expressions
- Know Pumping Lemma for Regular Languages
- Find Equivalence of two Regular Expressions

3.2 Introduction

In this chapter,we shall study about properties of regular languages. We have already defined regular languages in Chapter 1. "Whether every formal language is regular or not"?

In the next section we shall show the set of regular languages possess union, intersection, complement, concatenation and kleen closure property which help us to design DFA's for languages that are defined from regular languages.

Next, we shall see the construction of automata from the regular expressions, conversion of DFA to regular expression and vice versa.

In the last section, we shall study the necessary condition for the language to be regular, i.e., pumping lemma.

3.3 Regular Expressions

A regular expression is used to define a language.

A language is a set of **strings** involing **symbols** from some **alphabet.**

3.3.1 Primitive Regular Expressions

Following Regular expressions are primitive and Universal :

1. x, $\forall\, x \in \Sigma$ **of length 1.**
2. $\wedge$, the **empty string** and
3. ϕ, **no string** in particular language.

Now, we give formal definition of Regular expressions.

3.3.2 Definition

1. Every **primitive regular expression** is a **regular expression**, i.e., any terminal symbol, $\wedge$, and ϕ are regular expressions.
2. If R_1 and R_2 are two regular expressions than $R_1 + R_2$ (**Union** of two regular expressions) is a regular expressions.
3. If R_1 and R_2 are two regular then R_1R_2 (the **concatenation** of two regular expressions) is a regular expression.
4. If R is a regular expression than R^* (the iteration or closure of R) is a regular expression.
5. If R is regular expression, than (R) is also a regular expression.
6. The **complement** of a regular expression is a regular expression.
7. The **difference** of two regular expressions is a regular expression.

3.3.3 Meaning of notations

1. (x) : read as "**grouping of** x"
2. x^* : read as 'x **postfix star**' means **zero or more occurance** of the preceding regular expression. for example : a* means $\{\wedge, a, aa, aaa, ...\}$
3. xy : read as "Justaposition of x and y".
4. $x + y$: read as "**either** x **or** y".
5. x^+ : read as "**one or more occurance of** x".
6. $(\wedge + x)$ or $(x + \wedge)$: read as "**Zero or one occurance of** x".

3.3.4 Identity Rule for Regular Expressions

Suppose P, Q and R are three regular expressions, then following identities hold for the regular expressions :

1. $\phi + R = R$ (I_1)

2. $\phi R + R\phi = \phi$ (I_2)
3. $\wedge R = R\wedge = R$ (I_3)
4. $\wedge^* = \wedge$ and $\phi^* = \wedge$ (I_4)
5. $R + R = R$ (I_5)
6. $R^* R^* = R^*$ (I_6)
7. $RR^* = R^*R$ (I_7)
8. $(R^*)^* = R^*$ (I_8)
9. $\wedge + RR^* = R^* = \wedge + R^*R$ (I_9)
10. $(PQ)^*P = P(QP)^*$ (I_{10})
11. $(P + Q)^* = (P^* Q^*)^* = (P^* + Q^*)^*$ (I_{11})
12. $(P + Q)R = PR + QR$ and $R(P + Q) = RP + RQ$ (I_{12})

3.3.5 Construction of Regular Expressions

We can construct the regular expressions for the give languages using above identity rules and notations.

Example 3.1 : Find the regular expression for the language containing all strings having any number of 0's and 1's, except null string.

Solution : It is clear that the string contains at least **one 0's** or **one 1's**. Therefore, the regular expression will be:

$$(0 + 1)^+$$

Example 3.2 : Find the regular expression for the language accepting all the strings which ends with 00 over the set of input alphabet $\Sigma = \{0, 1\}$.

Solution : Since the string terminates with 00, it can have any number of 0's and 1's before 00.

$$\underbrace{\qquad}_{\text{zero or more occurance of 0's or 1's}} \quad \underbrace{0}_{} \quad \underbrace{0}_{}$$

The corrosponding regular expression will be

$$(0 + 1)^* 00$$

Example 3.3 : Find the regular expression for the strings which are starting with 0 and ending with 1 over the set of input alphabets $\Sigma = \{0, 1\}$.

Solution : As it is clear that the regular expression should start with 0 and end with 1.

$$\underbrace{0}_{} \quad \underbrace{\qquad}_{\text{any number of 0's or 1's}} \quad \underbrace{1}_{}$$

So, the required regulars expression is

$$0\ (0 + 1)^*\ 1.$$

Example 3.4 : Construct the regular expression for the strings in which any number of a's is followed by any number of b's is followed by any number of c's over the set of input alphabet Σ = {*a*, *b*, *c*}.

Solution :

zero or more occurance of a	zero or more occurance of b	zero or more occurance of c

The resulting regular expression will be

$$a^* b^* c^*$$

Example 3.5 : Construct the regular expression for the strings in which at least one a is followed by at least one b is followed by atleast one c over the set of input alphabet Σ = {*a*, *b*, *c*}.

Solution :

one or more occurance of a	one or more occurance of b	one or more occurance of c

The resulting regular expression will be

$$a^+ b^+ c^+$$

Example 3.6 : Find the regular expression which accepts all strings begining or ending with either 01 or 10.

Solution : The above problem can be divided into two subproblems. One, the string which begins with 01 or 10 and second, the strings which end with 10 or 01.

Let us find out the strings which **begin** with 01 or 10.

01 or 10	zero or more occurance of 0's or 1's.

The corrosponding regular expression will be

$$(01 + 10)\ (0 + 1)^* \qquad \text{....(1)}$$

Similarly, the regular expression for the strings ending with 10 or 01 will be

$$(0 + 1)^*\ (01 + 10) \qquad \text{....(2)}$$

Combining the regular expressions found in (1) and (2), we get,

$$\mathbf{(01 + 10)\ (0 + 1)^* + (0 + 1)^*\ (01 + 10)} \qquad \text{....(3)}$$

The regular expression found in equation (3) is the required regular expression.

Example 3.7 : Find the regular expression for the strings in which the third character from right is always 0 over the set of input alphabet Σ = {0, 1}.

Solution :

zero or more occurance of 0's or 1's	0	0 or 1	0 or 1

The regular expression will be $\mathbf{(0 + 1)^*\ 0\ (0 + 1)\ (0 + 1)}$

Example 3.8 : Find the regular expression for the set of all strings having at least two 1's over the set of input alphabets Σ = {0, 1}.

Solution : The resulting regular expression have restriction on the number of 1's, i.e., it will have at least two 1's. It can be developed as follows :

zero or more occurance of 0's or 1's — 1 — zero or more occurance or 0's or 1's — 1 — zero or more occurance of 0's or 1's

The corrosponding regular expression will be

$$(0 + 1)^* \, 1(0 + 1)^* \, 1 \, (0 + 1)^*$$

Example 3.9 : Find the regular expression representing the set of all strings over the set of input alphabets Σ = {a, b} in which the number of occurance of a is divisible by 3.

Solution : The regular expression will have the number of a's in multiple of 3. There is **no restriction on the number of b's**. The **position of a's and b's are also not fixed.**

zero or more b's — a — zero or more b's — a — zero or more b's — a — zero or more b's

zero or more occurance

The corrosponding regular expression will be as follows :

$$(b^* \, a \, b^* \, a \, b^* \, a \, b^*)^*$$

Example 3.10 : Find the regular expression for the set of all strings having atmost one pair of 0's or atmost one pair of 1's over the set of input alphabets Σ = {0, 1}.

Solution : The regular expression for the strings having at most one pair of 0's can be developed as follows :

Either it can have **no pair of 0's** or **one pair of 0's**.

The corrosponding regular expression will be

$$\underbrace{(1+01)^*}_{\text{no pair of 0's}} + \underbrace{(1+01)^*00(1+01)^*}_{\text{one pair of 0's}} \quad \ldots.(1)$$

Similarly, for the regular expression having atmost one pair of 1's.

$$\underbrace{(0+10)^*}_{\text{no pair of 1's}} + \underbrace{(0+10)^*11(0+10)^*}_{\text{one pair of 1's}} \quad \ldots.(2)$$

Combining regular expressions found in equation (1) and (2), we get,

$$(1 + 01)^* + (1 + 01)^* \, 00 \, (1 + 01)^* + (0 + 10)^* + (0 + 10)^* \, 11(0 + 10)^* \quad \ldots.(3)$$

The regular expression found in equation (iii) is the required regular expression.

Example 3.11 : Represent the following sets by regular expressions :

(a) {0, 1, 2}.

(b) $\{1^{2n+1} \mid n > 0\}$.

(c) $\{w \in \{a, b\}^* \mid w$ has only one $a\}$.

(d) The set of all strings over {0, 1} which has atmost two zeros.

(e) $\{a^2, a^5, a^8, ...\}$.

(f) $\{a^n \mid n$ is divisible by 2 or 3 or $n = 5\}$.

(g) The set of all strings over $\{a, b\}$ beginning and ending with a.

Solution : (a) Given : {1, 2, 3}.

The regular expression is :

$$(1 + 2 + 3)$$

(b) **Given :** $\{1^{2n+1} \mid n > 0\}$

The resutting set will have

$$\{111, 11111, 1111111, ...\}$$

The regular exforession is :

$$1(11)^+$$

(c) **Given :** $\{w \in \{a, b\}^* \mid w$ has only one $a\}$

Since, it can have only one **a** but it can have any number of **b's** before and after a.

$\therefore$ The regular expression will be as follows.

$$b^* ab^*.$$

(d) **Given :** $\Sigma = \{0, 1\}$

To find : The regular expression which have at most two zero's,

i.e., it can have

(i) No 0's

(ii) One 0's

(iii) Two 0's

$$\underbrace{1^*}_{\text{No 0's}} \text{ or } \underbrace{1*01*}_{\text{One 0's}} \text{ or } \underbrace{1*01*01*}_{\text{two 0's}}$$

The corrosponding regular expression will be:

$$1^* + 1^* 01^* + 1^* 01^* 01^*$$

(e) **Given :** $\{a^2, a^5, a^8, ...\}$

The resulting regular expression is as follows :

$$aa(aaa)^+$$

(f) **Given** : $\{a^n \mid n$ is divisible by 2 or 3 or $n = 5\}$

The corrosponding regular expression will be

$$(aa)^* + (aaa)^* + aaaaa$$

(g) **Given** : $\Sigma = \{a, b\}$

To find : Set of all strings begining and ending with a.

The corrosponding regular expression wil lbe as follows :

$$a(a + b)^*a.$$

Example 3.12 : Describe in English the language represented by the following regular expressions

(a) $(a + ab)^*$

(b) $(b^* (aaa)^* b^*)^*$

(c) $a(a + b)^* ab$

(d) $a^* b + b^*a$

(e) $(aa + b)^* (bb + a)^*$.

Solution : (a) Given regular expression = $(a + ab)^*$

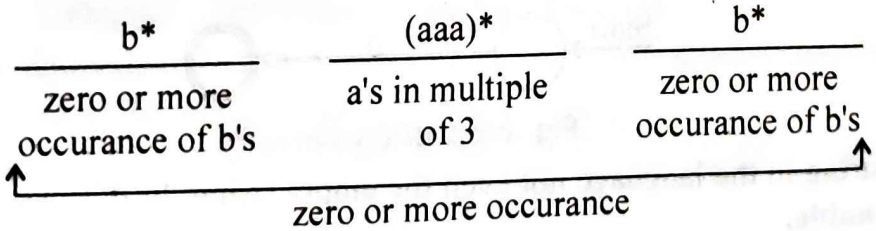
The set of strings generated by the above regular expression is

$$\{a, aba, abab, aab, aaa \ldots.\}$$

The above regular expression represents the strings **begining with a but not having two consecutive b's.**

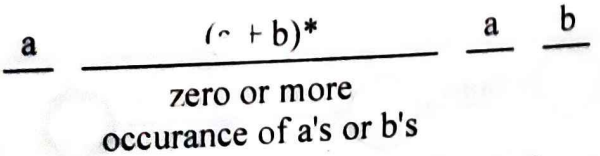
(b) Given regular expression = $(b^* (aaa)^* b^*)^*$

The above regular expression can be divided as follows :



The above regular expression represents the strings in which a's appear in multiple of 3 and there is no restriction on number of b's.

(c) Given regular expression = $a(a + b)^* ab$



The above regular expression represents the **strings begining with a and ending with** ab.

(d) Given regular expression $RE = a^*b + b^*a$.

$\underbrace{a^*}_{\text{zero or more occurance of a's}}\ b$ OR $\underbrace{b^*}_{\text{zero or more occurance of b's}}\ a$

The above regular expression represents the set of strings having either **string of a's followed by one b or string of b's followed by one *a*.**

(e) Given regular expression $RE = (aa + b)^* (bb + a)^*$

The above regular expression can be represented as follows :

$$\underbrace{(aa + b)^*}_{W_1}\ \underbrace{(bb + a)^*}_{W_2}$$

The above regular expression represents the set of all strings of the form W_1W_2 in which a's appear in pairs i W_1 and b's appear in pairs in W_2.

3.4 Regular Expressions And Finite Automata

For any regular expression, there is an equivalent non-deterministic finite automata (NDFA) and hence, deterministic finite automata (DFA), that accepts the same set of language and vice versa.

3.4.1 Primitive Regular Expressions and Equivalent NDFA

1. $x \in \Sigma$

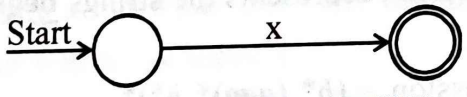


Fig. 3.1 : NDFA for $x \in \Sigma$.

2. $\wedge$: The language containing empty string.

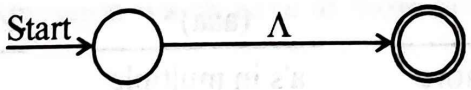


Fig. 3.2 : NDFA for $\wedge$.

3. ϕ : No string in the language not even the empty string. In this case, the **final**/accepting state is **not reachable.**

Fig. 3.3 : NDFA for ϕ.

or

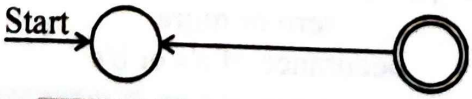


Fig. 3.4 : NDFA for ϕ.

3.4.2 Other Regular Expressions and Equivalent NDFA

As we know, every primitive regular expression is a regular expression and by applying following operations, we get the regular expressions.

- Grouping
- Justaposition
- Union
- Kleen Star

In this section, we shall see, how these operations can result into equivalent NDFA.

1. Grouping : As we know that the basic primitive regular expression x and regular expression (x) denote the same set of languages, thus, the equivalent NDFA, for the regular expressions (x) is also same to the equivalent NDFA of the primitive regular expression.

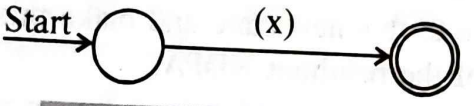


Fig. 3.5 : NDFA for (x).

2. Concatenation (Justaposition) : Let a and b be the two primitive regular expressions, then the corrosponding NDFA for the regular expressions a and b are as follows :

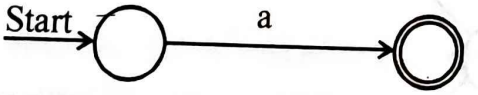


Fig. 3.6 : NDFA for regular expression *a*.

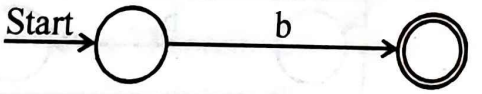


Fig. 3.7 : NDFA for regular expression *b*.

Now, to find the equivalent NDFA for the concatenation operation on these primitive regular expressions, written as ab, perform the following steps.

(i) Apply the null move ($\wedge$-move) from final state of first NDFA to the start state of second NDFA.

(ii) Make the start of the NDFA of first primitive regular expression as the start state of the resultant NDFA.

(iii) Make the final state of the NDFA of second primitive regular expression as the final state of the resultant NDFA.

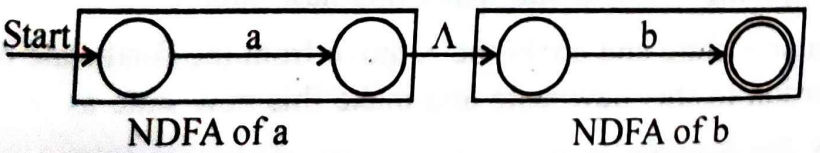


Fig. 3.8 : NDFA for *ab*.

3. Union : Let us, again, assume, that ***a*** and ***b*** are two primitive regular expressions then equivalent NDFA's for these regular expressions *a* and *b* are as follows :

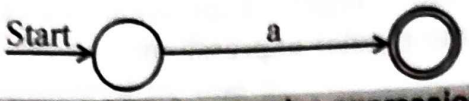


Fig. 3.9 : NDFA for regular expression *a*.

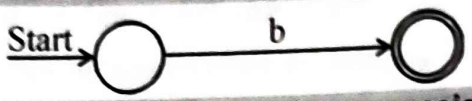


Fig. 3.10 : NDFA for regular expression *b*.

Now, to find an equivalent NDFA for the union operation on these primitive regular expressions, written as ***a + b***, perform the following operation :

(i) Take a new state which can go to the start state of both the NDFA by applying ∧-move and make it the **new start state** of the resultant NDFA.

(ii) Take a new state and from the final states of both the primitive regular expressions, apply the ∧-move to this new state and make this new state as the new final state/ accepting state of the resultant NDFA.

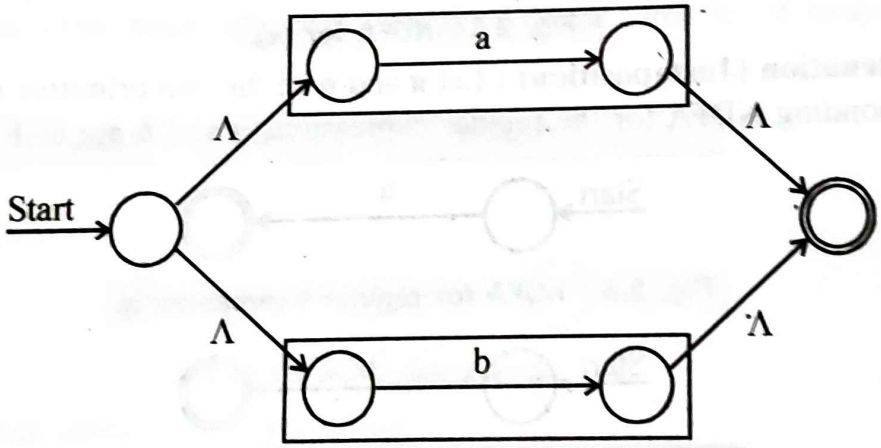


Fig. 3.11 : Resultant NDFA for *a + b*.

4. Kleen star (Postfix star) : Let us suppose that a is the primitive regular expression, then the equivalent NDFA is

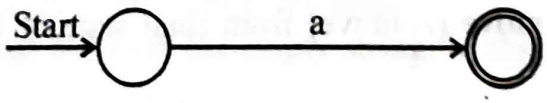


Fig. 3.12 : NDFA of a.

To find the equivalent NDFA for the Kleen star of a, written as a^*, perform the following steps :

(i) Take a new state, which can go to the start state of the primitive regular expression ***a*** by applying ∧-move and make this new state as the new start state.

(ii) Take a new state and apply the ∧-move from the final state of the primitive regular expression to this new state and make this new state as the **new final state**.

(iii) Apply the ∧-move from the final state of the primitive regular expression to the **initial state** of the primitive regular expression.

(iv) Apply the ∧-move from **new start state** to the **new final state.**

The resultant NDFA for the regular expression a* is shown below :

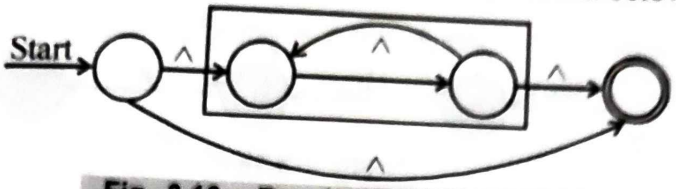


Fig. 3.13 : Resultant NDFA for a*.

3.4.3 Construction of Regular Expression from Finite Automata

In this section, we shall see one of the important theorem which will help us in recognizing the string by the given finite automata.

Arden's Theorem

Let P and Q be the two regular expressions over some input alphabet Σ. If P does not contain ∧, then the following equation in R, i.e.,

$$R = Q + RP \quad \text{....(1)}$$

has a unique solution (one and only one solution) given by

$$R = QP^* \quad \text{....(2)}$$

Proof :

Putting value of R from equation (2) in equation (1), we get,

$$Q + (QP^*)\,P = Q(\wedge + P^*P)$$

$$= QP^* \text{ (from Identity } I_9\text{)}.$$

Note : We can apply ARDEN's Theorem only when :

1. The finite automata / transition system **does not contain** any **∧-move.**
2. The given finite automata contains only one start state.

Algebric method to find regular expression recognised by finite automata

Let $q_1, q_2, q_3 \ldots\ldots q_n$, be the set of states where q_1 is the start state of the given finite automata and α_{ij} denote the regular expression from q_i to q_j where, $1 \le i \le n$ and $1 \le j \le n$

Then, we get the following equations :

$$q_1 = q_1\alpha_{11} + q_2\alpha_{21} + \ldots . q_n\alpha_{n1} + \Lambda \quad \text{....(1)}$$

$$q_2 = q_1\alpha_{12} + q_2\alpha_{22} + \ldots . q_n\alpha_{n2} \quad \text{....(2)}$$

$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$$

$$q_m = q_1\alpha_{1m} + q_2\alpha_{2m} + \ldots . q_n\alpha_{nm} \quad \text{....(m)}$$

Solve the above set of equations by applying substitutions and ARDEN's Theorem.

∧ is added in equation q_1 because it is the start state.

Example 3.13 : Design the regular expression corrosponding to the DFA shown in Fig. 3.14 by Arden's theorem.

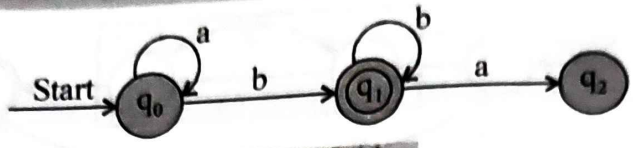


Fig. : 3.14

Solution : The given DFA does not contain any ∧-move and has only one initial state, we can apply Arden's theorem.

State = source state of input × input coming to it.

$$q_0 = q_0a + \wedge \quad (\because q_0 \text{ is the start state, } \therefore \wedge \text{ is added}) \quad \text{....(1)}$$

Similarly, $$q_1 = q_0b + q_1b \quad \text{....(2)}$$

Arranging equation (i), we get,

$$q_0 = \wedge + q_0a \quad \text{....(3)}$$

Comparing equation (3) with $R = Q + RP$ which gives R = QP*

Assuming $R = q_0$, $Q = \wedge$ and $P = a$

We get, $$q_0 = \wedge \cdot a^*$$

$$q_0 = a^* \quad (\because \wedge \cdot R = R) \quad \text{....(4)}$$

Substituting value of q_0 in equation (2), we get,

$$q_1 = a^*b + q_1b \quad \text{....(5)}$$

Comparing equation (5) with $R = Q + RP$ which gets reduced to $R = QP^*$ and assuming $R = q_1$, $Q = a^* b$ and $P = b$.

We get, $$q_1 = a^* b \cdot b^*$$

$\therefore$ $$q_1 = a^* \cdot b^+$$ (As $RR^* = R^+$. Here, it is b)

Result : q_1 is the final state hence, the regular expression for the given DFA is $a^* \cdot b^+$.

Example 3.14 : Design the regular expression corresponding to the DFA shown in Fig. 3.15 by Arden's theorem. (A-2, 2C-DL)

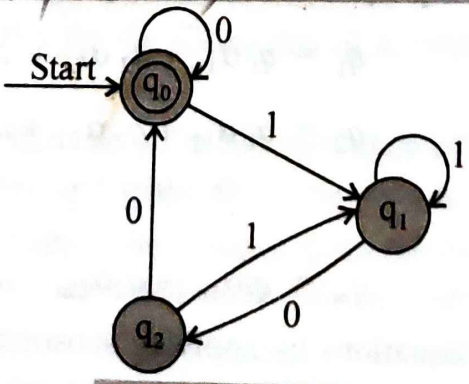


Fig. : 3.15

Solution : State = source state of input × input coming to it.

$$\therefore \quad q_0 = q_0 0 + q_2 0 + \wedge \quad (q_0 \text{ is the start state}) \qquad \text{....(1)}$$

Similarly,

$$q_1 = q_0 1 + q_1 1 + q_2 1 \qquad \text{....(2)}$$

$$q_2 = q_1 0 \qquad \text{....(3)}$$

Substituting value of q_2 from equation (3) in equation (2), we get,

$$q_1 = q_0 1 + q_1 1 + q_1 01 \qquad \text{....(4)}$$

$$q_1 = q_1(1 + 01) + q_0 1 \qquad \text{....(5)}$$

Comparing equation (5) with $R = Q + RP$ which gets reduced to $R = QP^*$ and Assuming $R = q_1$, $Q = q_{01}$, $P = (1 + 01)$.

we get, $$q_1 = q_0 1(1 + 01)^* \qquad \text{....(6)}$$

Substituting value of q_2 from equation (3) in equation (1), we get,

$$q_0 = q_0 0 + q_1 00 + \wedge \qquad \text{....(7)}$$

Putting value of q_1, from equation (6) in equation (7), we get,

$$q_0 = q_0 0 + q_0 1(1 + 01)^* 00 + \wedge$$

$$q_0 = q_0(0 + 1(1 + 01)^* 00) + \wedge \qquad \text{....(8)}$$

Again comparing equation (8) with $R = Q + RP$ and

Assuming, $$R = q_0, \; Q = \wedge, \; P = 0 + 1(1 + 01)^* 00$$

Which gets reduced to $R = QP^*$.

$$q_0 = \wedge(0 + 1(1 + 01)^* 00)^*$$

$$\therefore \quad q_0 = (0 + 1(1 + 01)^* 00)^* \qquad (\wedge \cdot R = R)$$

Result : Since, q_0 is the **final state** hence, the regular expression for the given DFA is

$$(0 + 1(1 + 01)^* 00)^*$$

Example 3.15 : Design the regular expression corresponding to the DFA shown in Fig. 3.16 by Arden's theorem.

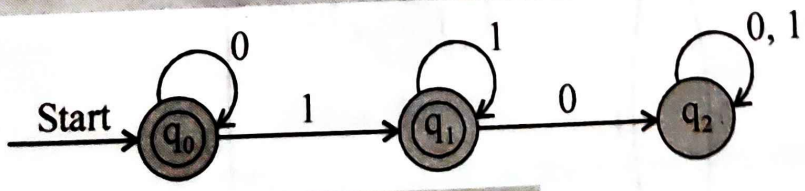


Fig. 3.16

Solution : State = Source state of input × input coming to it.

$$q_0 = q_0 0 + \wedge \quad (q_0 \text{ is the start state}) \qquad \text{....(1)}$$

Similarly,

$$q_1 = q_0 1 + q_1 1 \quad \text{....(2)}$$

$$q_2 = q_1 0 + q_2 (0 + 1) \quad \text{....(3)}$$

For getting the regular expression, we have to solve the **final state**. Since, we have two final states q_0 and q_1, we have to solve for both q_0 and q_1.

Comparing equation (1) with $R = Q + RP$ which gets reduced to $R = QP^*$ with $Q = \wedge$, $P = 0$ and $R = q_0$

we get,

$$q_0 = \wedge 0^*$$

$$q_0 = 0^* \qquad (\wedge \cdot R = R) \quad \text{....(4)}$$

Substituting the value of q_0 from equation (4) to equation (2), we get,

$$q_1 = 0^* 1 + q_1 1 \quad \text{....(5)}$$

Again, comparing equation (5) with

$$R = Q + RP$$

which gets reduced to $R = QP^*$

with $R = q_1$, $Q = 0^* 1$ and $P = 1$

we get,

$$q_1 = 0^* 1 (1)^* \quad \text{....(6)}$$

Since, this regular expression contains **two final states**, namely, $\mathbf{q_0}$ and $\mathbf{q_1}$, therefore, the final regular expression is given by

$$R = q_0 + q_1 \quad \text{....(7)}$$

Putting value of q_0 and q_1 from equations (4) and (6) in equation (7), we get,

$$R = 0^* + 0^* 1 (1)^*$$

$$\boldsymbol{R = 0^* + 0^* 1^+}$$

Example 3.16 : Design the regular expressions corresponding to the DFA shown in Fig. 3.17 by Arden's theorem. JL-AL-I

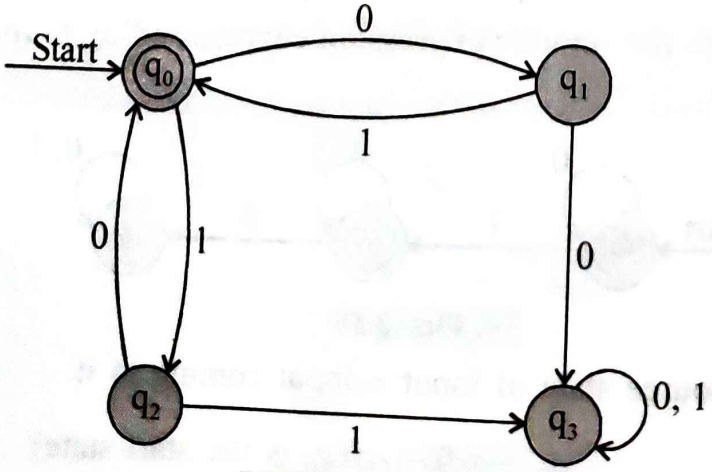


Fig. 3.17

Solution : State = source state of input × input coming to it

$$q_0 = q_1 1 + q_2 0 + \wedge \quad (q_0 \text{ is the start state}) \qquad \ldots(1)$$

Similarly,

$$q_1 = q_0 0 \qquad \ldots(2)$$

$$q_2 = q_0 1 \qquad \ldots(3)$$

$$q_3 = q_1 0 + q_2 1 + q_3(0 + 1) \qquad \ldots(4)$$

For getting the regular expression, we have to solve the **final state** which is q_0. putting the value of q_1 and q_2 from equation (2) and equation (3) into equation (1), We get,

$$q_0 = q_0 01 + q_0 10 + \wedge$$

$$q_0 = q_0(01 + 10) + \wedge \qquad \ldots(5)$$

comparing equation (5) with

$$R = RP + Q$$

which get reduced to

$$R = QP^*$$

with $R = q_0, \quad P = (01 + 10)$ and $Q = \wedge$,

we get, $q_0 = \wedge(01 + 10)^*$

$$q_0 = (01 + 10)^* \qquad (\wedge R = R)$$

Result : Since, q_0 is the **final state,** hence, the regular expression for the given DFA is **(01 + 10) *.**

Example 3.17 : Design the regular expressions corresponding to the DFA shown in Fig. .18 by Arden's theorem.

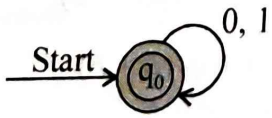


Fig. 3.18

Solution : There is only one state and also it is the initial and final state.

State = Source state of input × input coming to it.

$$q_0 = q_0 0 + q_0 1 + \wedge \qquad \ldots(1)$$

$$q_0 = q_0(0 + 1) + \wedge \qquad \ldots(2)$$

Comparing equation (2) with $R = Q + RP$, with

$$R = q_0, \quad P = (0 + 1) \text{ and } Q = \wedge$$

we get,

$$q_0 = \wedge(0 + 1)^*$$

$$q_0 = (0 + 1)^* \qquad (\wedge \cdot R = R)$$

Result : Since, q_0 is the **final state** hence, the regular expression for the given DFA is **(0 + 1)***

Example 3.18 : Design the regular expressions corresponding to the DFA shown in Fig. 3.19 by Arden's theorem. 2C-AL-I

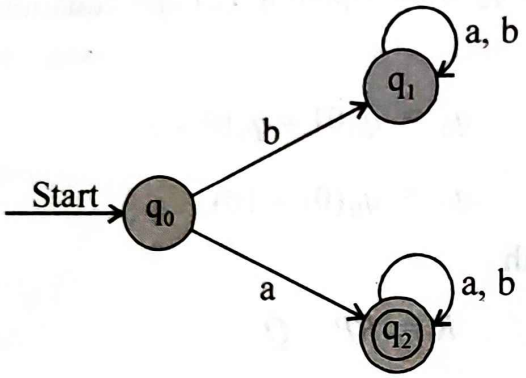


Fig. 3.19

Solution : State = source state of input × input coming to it

$$q_0 = \wedge \qquad \ldots(1)$$

(q_0 is the start state and no incoming edge is associated with it)
Similarly,

$$q_1 = q_1(a + b) + q_0 b \qquad \ldots(2)$$

and

$$q_2 = q_0 a + q_2(a + b) \qquad \ldots(3)$$

putting value of q_0 from equation (1) into equation (3), we get,

$$q_2 = \wedge a + q_2(a + b)$$

$$q_2 = a + q_2(a + b) \qquad (\wedge \cdot R = R)$$

comparing this equation with

$R = Q + RP$ which gets reduced to $R = QP^*$

with

$$R = q_2, \quad Q = a, \quad P = (a + b)$$

We get,

$$q_2 = a(a + b)^*$$

Result : q_2 is the **final state** hence regular expression for the given DFA is $a(a + b)^*$.

Example 3.19 : Design the regular expressions corresponding to the DFA shown in Fig. 3.20 by Arden's theorem.

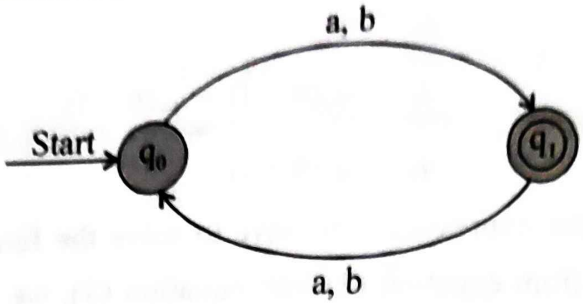


Fig. 3.20

Solution :

State = Source state of input × input coming to it.

$$q_0 = q_1a + q_1b + \wedge \qquad (q_0 \text{ is the start state})$$

$$q_0 = q_1(a + b) + \wedge \qquad \text{....(1)}$$

Similarly, $$q_1 = q_0(a + b) \qquad \text{....(2)}$$

for getting the regular espressions we have to solve the final state which is q_1.

putting the value of q_0 from equation (1) in equation (2)

$$q_1 = ((a + b)q_1 + \wedge)(a + b)$$

$$q_1 = (a + b)(a + b)q_1 + \wedge(a + b)$$

$$q_1 = (a + b)(a + b)q_1 + (a + b) \qquad (\wedge R = R)$$

comparing this equation with $R = Q + RP$

which gets reduced to $R = QP^*$

with $R = q_1,\ Q = (a + b),\ P = (a + b)(a + b)$

We get, $$q_1 = (a + b)((a + b)(a + b))^*$$

Result : q_1 is the **final state**, hence, the regular expression for the given DFA is $(a + b)((a + b)(a + b))^*$.

Example 3.20 : Design the regular expressions corresponding to the DFA shown in Fig. 3.21 by Arden's theorem.

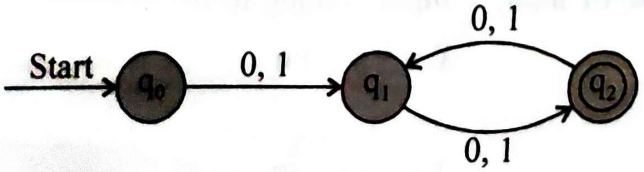


Fig. 3.21

Solution :

State = source state of input × incoming input to it.

$q_0 = \wedge$ (q_0 is the start state) ....(1)

$q_1 = q_0(0 + 1) + q_2(0 + 1)$ ....(2)

and $q_2 = q_1(0 + 1)$ ....(3)

For getting the regular expressions we have to solve the final state which is q_2.

Putting value of q_1 from equation (2) into equation (3), we get,

$$q_2 = (0 + 1)(q_0(0 + 1) + q_2(0 + 1)) \quad ....(4)$$

Putting value of q_0 from equation (1) into equation (4), we get,

$$q_2 = (0 + 1)((0 + 1) + q_2(0 + 1))$$

$$q_2 = (0 + 1)(0 + 1) + q_2(0 + 1)(0 + 1) \quad ....(5)$$

comparing with equation (5) with

$$R = Q + RP$$

which gets reduced to $R = QP^*$

with $R = q_2$, $Q = (0 + 1)(0 + 1)$ and

$P = (0 + 1)(0 + 1)$

We get, $q_2 = (1 + 0)(1 + 0)((1 + 0)(1 + 0))^*$

Result : q_2 is the **final state** hence the regular expressions for the given DFA is **(1 + 0) (1 + 0) ((1 + 0) (1 + 0))*.**

Example 3.21 : Design the regular expression corresponding to the DFA shown in Fig. 3.22 by Arden's theorem.

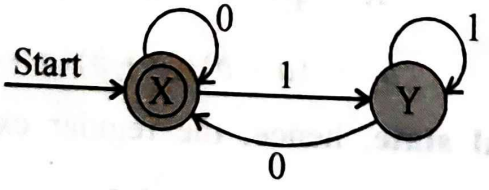


Fig. 3.22

Solution :

State = source state of input × input coming to it.

$X = X0 + Y0 + \wedge$ (X is the start state) ....(1)

Similarly,

$$Y = X1 + Y1 \quad ....(2)$$

Comparing this equation with $R = Q + RP$ which gets reduced to $R = QP^*$, with $R = Y$, $Q = X1$ and $P = 1$ we get,

$$Y = X11^*$$

$$Y = X1^+ \qquad (RR^* = R^+) \qquad \text{....(3)}$$

putting the value of Y from equation (3) into equation (1) we get

$$X = X0 + X1^+0 + \wedge$$

$$X = X((0 + 1^+0)) + \wedge \qquad \text{....(4)}$$

comparing equation (4) with $R = Q + RP$

Which gets reduced to $R = QP^*$

with $R = X$, $Q = \wedge$ and $P = (0 + 1^+0)$, we get,

$$X = \wedge(0 + 1^+0)^*$$

$$X = (0 + 1^+ 0)^* \qquad (\wedge R = R)$$

Result : X is the **final state**, hence, the regular expressions for the given DFA is $(0 + 1^+ 0)^*$.

Example 3.22 : Design the regular expression corresponding to the DFA shown in Fig. 3.23 by Arden's theorem.

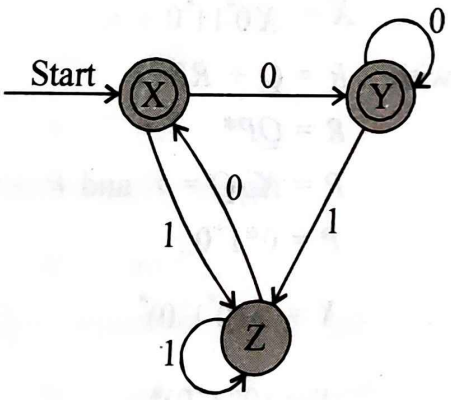


Fig. 3.23

Solution :

State = source state of input × input incoming to it.

$$X = Z0 + \wedge \qquad (X \text{ is the start state}) \qquad \text{....(1)}$$

Similarly,

$$Y = X0 + Y0 \qquad \text{....(2)}$$

$$Z = X1 + Y1 + Z1 \qquad \text{....(3)}$$

First, we will solve the equation (2)

$$Y = X0 + Y0$$

Comparing equation (2) with $R = Q + RP$ with

$R = Y$, $Q = X0$, and $P = 0$

Which get reduced to $R = QP^*$.

$$Y = X00^* \qquad \text{....(4)}$$
$$Y = X0^+ \qquad (RR^* = R^+)$$

putting the value of Y from equation (4) into equation (3)

$$Z = X1 + X0^+1 + Z1 \qquad \text{....(5)}$$
$$Z = X(1 + 0^+1) + Z1$$

comparing the equation (5) with

$$R = Q + RP$$

which gets reduced to $R = QP^*$

with $R = Z$, $Q = X(1 + 0^+1)$ and $P = 1$

$$Z = X(1 + 0^+1)1^*$$
$$Z = X((\wedge + 0^+)1)1^*$$
$$Z = X0^*\,11^* \qquad (\wedge + R^+ = R^*) \qquad \text{....(6)}$$

putting the value of Z from equation (6) into equation (1), we get,

$$X = X0^*11^*0 + \wedge \qquad \text{....(7)}$$

comparing equation (7) with $R = Q + RP$

which gets reduced to $R = QP^*$

with $R = X$, $Q = \wedge$ and $P = 0^*11^*0$

$$P = 0^*1^+0 \qquad (RR^* = R^+)$$

we get, $X = \wedge(0^*1^+0)^*$

$$X = (0^*1^+0)^* \qquad (\wedge R^* = R^*)$$

For getting the regular expression we have to solve the **final states**. Since, there are **two final** states namely, X and Y, in given DFA, we have to solve both the states X and Y.

The final regular expression is given by

$$R = X + Y$$
$$R = (0^*1 + 0)^* + (X0^+)$$
$$R = (0^*1 + 0)^* + ((0^*1 + 0)^*0^+)$$

Result : $\because$ X and Y both are the final states

$\therefore$ the regular expression for the given DFA is

$$(0^*\,1^+\,0)^* + (0^*\,1^+\,0)^*\,0^+.$$

Example 3.23 : Design the regular expression corrosponding to the DFA shown in Fig. 3.24.

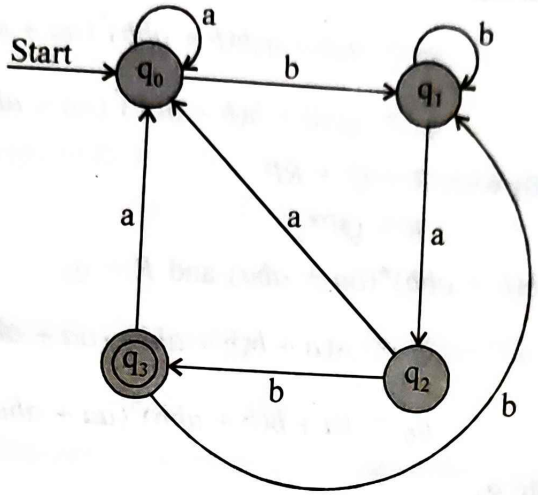


Fig. 3.24.

Solution : As q_0 is the start state, the equation of q_0 will be

$$q_0 = q_0a + q_2a + q_3a + \wedge \quad \text{....(1)}$$

Similarly,

$$q_1 = q_0b + q_1b + q_3b \quad \text{....(2)}$$

$$q_2 = q_1a \quad \text{....(3)}$$

$$q_3 = q_2b \quad \text{....(4)}$$

putting the value of q_2 in equation (4), we get,

$$q_3 = q_1ab \quad \text{....(5)}$$

putting value of equation (5) in equation (2), we get,

$$q_1 = q_0b + q_1b + q_1abb \quad \text{....(6)}$$

$$q_1 = q_1(b + abb) + q_0b$$

$$q_1 = q_0b + q_1(b + abb) \quad \text{....(7)}$$

Comparing equation (7) with $R = Q + RP$ which gets reduced to QP^* with $Q = q_0$, $P = (b + abb)$ and $R = q_1$

we get,

$$q_1 = q_0b(b + abb)^* \quad \text{....(8)}$$

putting value of q_2 and q_3 from equation (3) and (4) in equation (1), we get,

$$q_0 = q_0a + (q_1a)a + (q_1ab)a + \wedge$$

$$q_0 = q_0a + q_1(aa + aba) + \wedge \quad \text{....(9)}$$

putting value of q_1 from equation (8) in equation (9), we get,

$$q_0 = q_0a + q_0b(b + abb)^*(aa + aba) + \wedge$$

$$q_0 = q_0(a + b(b + abb)^*(aa + aba) + \wedge \quad \text{....(10)}$$

comparing equation (10) with $R = Q + RP$

Which gets reduced to $R = QP^*$,

with $Q = \wedge$, $P = a + b(b + abb)^*(aa + aba)$ and $R = q_0$

we get,

$$q_0 = \wedge(a + b(b + abb)^*(aa + aba))^*$$

$$q_0 = (a + b(b + abb)^*(aa + aba))^* \quad (\wedge R = R) \ldots(11)$$

Now, we solve for state q_3

$$q_3 = q_1ab$$

Putting value of q_1 from equation (8), we get,

$$q_3 = q_0b(b + abb)^*ab$$

Putting value of q_0 from equation (11), we get,

$$q_3 = (a + b(b + abb)^*(aa + aba))^*b(b + abb)^*ab$$

As q_3 is the **final state**

$\therefore$ The regular expression for the finite automata is as follows.

$$RE = (a + b(b + abb)^*(aa + aba))^*b(b + abb)^*ab$$

Example 3.24 : Find the regular expression corrosponding to the transition diagram given below.

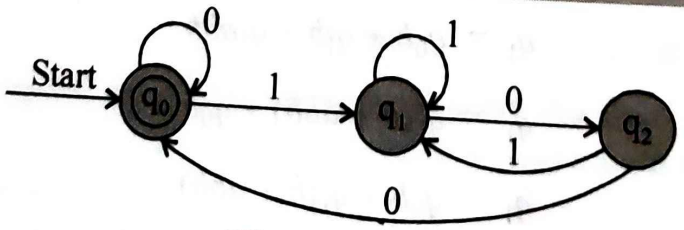


Fig. 3.25

Solution : As state q_0 is the start state, the equation for q_0 will be

$$q_0 = q_00 + q_20 + \wedge \quad \text{....(1)}$$

Similarly, equation for q_1 and q_2 will be :

$$q_1 = q_01 + q_11 + q_21 \quad \text{....(2)}$$

$$q_2 = q_1 0 \quad (3)$$

putting value of q_2 in equation (2), we get,

$$q_1 = q_0 1 + q_1 1 + q_1 01$$

$$q_1 = q_1(1 + 01) + q_0 1 \quad ....(4)$$

Comparing equation (4) with $R = Q + RP$ which gets reduced to QP^* with

$$q = q_0 1,\ R = q_1 \text{ and } P = (1 + 01)$$

We get,

$$q_1 = q_0 1(1 + 01)^* \quad ....(5)$$

Putting value of q_1 obtained in equation (5) in equation (3), we get,

$$q_2 = q_0 1(1 + 01)^* 0 \quad ....(6)$$

Putting value of q_2 obtained in equation (6) in equation (1), we get,

$$q_0 = q_0 0 + q_0 1(1 + 01)^* 00 + \wedge$$

$$q_0 = q_0(0 + 1(1 + 01)^* 00) + \wedge \quad ....(7)$$

As $R = Q + RP$ gets reduced to QP^*, comparing equation (7) with

$$R = Q + RP,$$

which gets reduced to $R = QP^*$

with

$$Q = \wedge$$

$$R = q_0$$

$$P = (0 + 1(1 + 01)^*00)$$

$$q_0 = \wedge(0 + 1(1 + 01)^* 00)^*$$

$$q_0 = (0 + 1(1 + 01)^* 00)^*$$

Result : As q_0 is the final state, the regular expression for the finite automata is :

$$\boldsymbol{RE = (0 + 1(1 + 01)^*00)^*}$$

3.4.4 Construction of Finite Automata From Regular Expression

To find out the DFA equivalent to the given regular expression, we will use the subset method, It involves following steps :

1. Construct the transition graph (transition system) from the given regular expression using null-move ($\wedge$-move), i.e., Find NDFA from Regular expression.
2. Construct the transition table from the graph in step 1 and construct the equivalent DFA., i.e., Find DFA from NDFA.

Note : Regular expression → Non Determinstic Finite Automata → Determinstic Finite Automata.

Example 3.25 : Design the Non-determintic finite automata for the regular expression $b + ba^*$.

Solution : Given :

$$RE = b + ba^*$$

It can be divided into RE_1 and RE_2 as follows :

$$RE_1 = b$$

$$RE_2 = ba^*$$

The regular expression RE_2 can be further divided into RE_3 and RE_4 as follows :

$$RE_3 = b$$

$$RE_4 = a^*$$

The resulting regular expression will be

$$RE = RE_1 + RE_3 \cdot RE_4$$

Now, we draw the NDFA for RE_1, RE_3 and RE_4.

NDFA for $RE_1 = b$:

Fig. 3.26 : Finite Automata for RE_1.

NDFA for $RE_3 = b$:

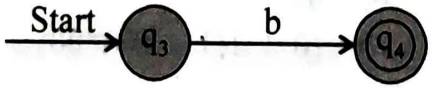


Fig. 3.27 : Finite Automata for RE_3.

NDFA for $RE_4 = a^*$:

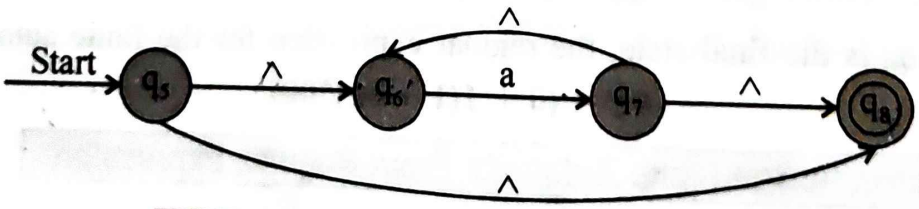


Fig. 3.28 : Finite Automata for RE_4.

The NDFA for $RE_2 = RE_3 \cdot RE_4$:

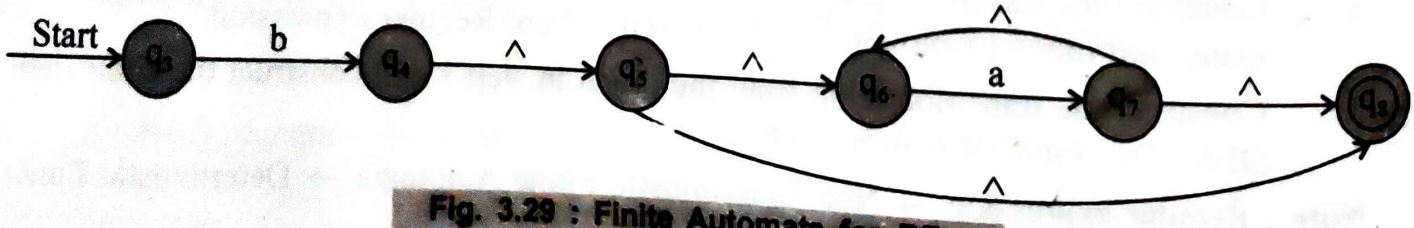


Fig. 3.29 : Finite Automata for RE_2.

The resultant NDFA for regular expression $RE = RE_1 + RE_2 = b + ba^*$ is drawn below.

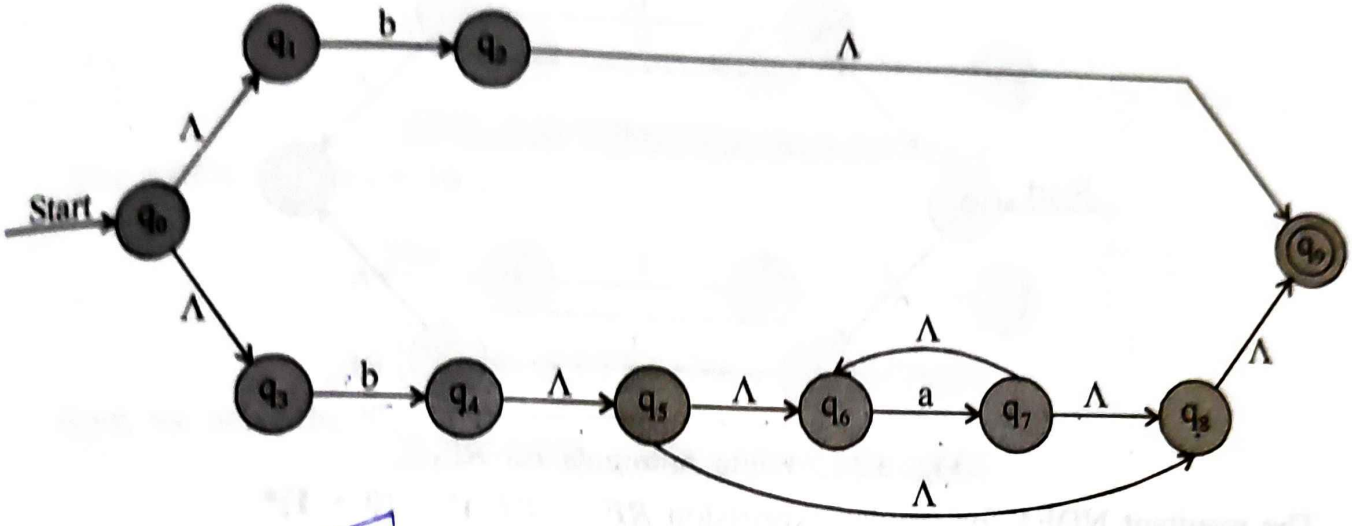


Fig. 3.30 : Finite Automata for $b + ba^*$.

Example 3.26 : Design the NDFA for the regular expression $(0 + 1)^*$.

Solution : Given

$$RE = (0 + 1)^*$$

It can be divided into smaller regular expressions

$$RE = (RE_1)^*$$

Regular expression RE_1 can further be divided as follows :

$$RE_1 = RE_2 + RE_3 \text{ where,}$$

$$RE_2 = 0 \text{ and}$$

$$RE_3 = 1$$

Now, we draw the NDFA for RE_2 and RE_3 :

The NDFA for $RE_2 = 0$:

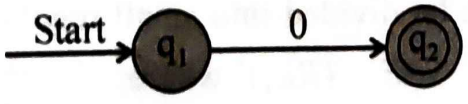


Fig. 3.31 : Finite Automata for RE_2.

The NDFA for $RE_3 = 1$:

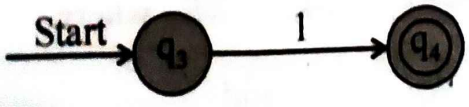


Fig. 3.32 : Finite Automata for RE_3.

The NDFA for $RE_1 = RE_2 + RE_3 = 0 + 1$

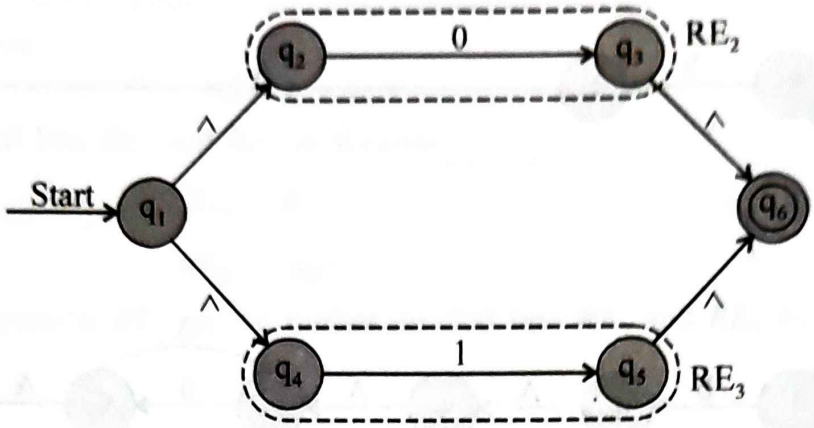


Fig. 3.33 : Finite Automata for RE_1.

The resultant NDFA for regular expression $RE = (RE_1)^* = (0 + 1)^*$

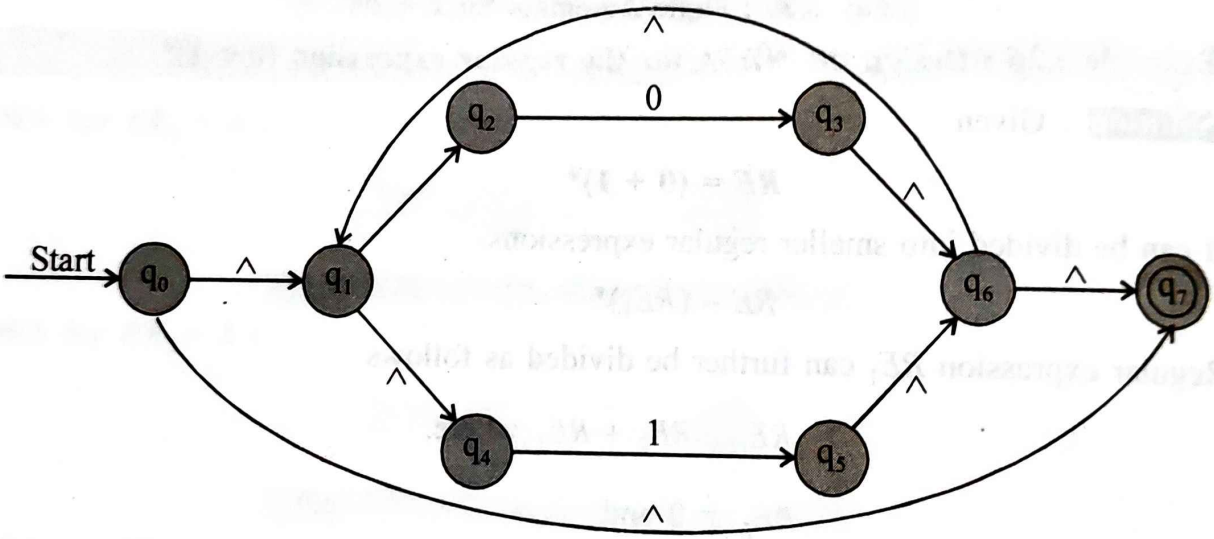


Fig. 3.34 : Finite Automata for (0 + 1)*.

Example 3.27 : Construct the NDFA for the regular expression $(01 + 10)^+$.

Solution : Given :

$$RE = (01 + 10)^+$$

This regular expression can be divided into small regular expressions as follows

$$RE = (RE_1)^+ \text{ where,}$$

$$RE_1 = (01 + 10)$$

RE_1 can be further divided into smaller expressions as follows :

$$RE_1 = RE_2 + RE_3 \text{ where,}$$

$$RE_2 = 01 \text{ and}$$

$$RE_3 = 10$$

Now, we draw the NDFA for RE_2 and RE_3

The NDFA for $RE_2 = 01$:

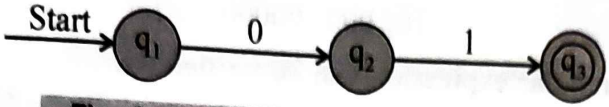


Fig. 3.35 : Finite Automata for RE_2.

The NDFA for $RE_3 = 10$:

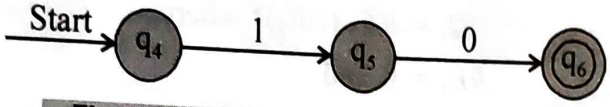


Fig. 3.36 : Finite Automata for RE_3.

Now, we draw the NDFA for $RE_1 = RE_2 + RE_3 = 01 + 10$:

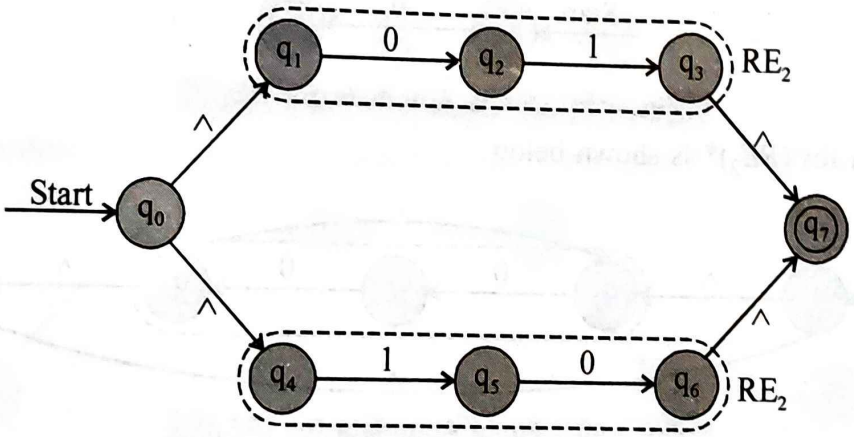


Fig. 3.37 : Finite Automata for RE_1.

The resultant NDFA for regular expression $RE = (RE_1)^+ = (01 + 10)^+$ is drawn below.

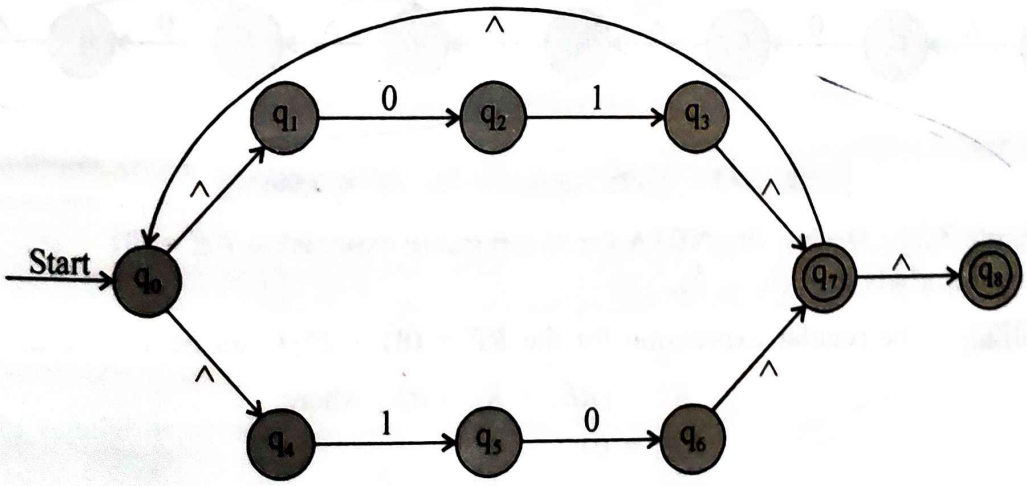


Fig. 3.38 : Finite Automata for $RE = (01 + 10)^+$.

Example 3.28 : Design the NDFA for the strings having odd number of 0's over the set of input alphabets $\Sigma = \{0\}$.

Solution : First, we draw the regular expression for the strings having odd number of 0's over $\Sigma = \{0\}$. It can have the following elements :

$$\{0, 000, 00000, ...\}$$

The corrosponding regular expression can be written as

$$RE = 0(00)^*$$

To draw the NDFA for $RE = 0(00)^*$, we divide the regular expression RE as follows :

$$RE = RE_1 (RE_2)^* \text{ where,}$$

$$RE_1 = 0 \text{ and}$$

$$RE_2 = 00$$

The NDFA for RE_1 can be drawn as follows :

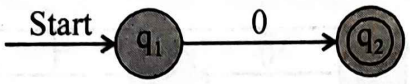


Fig. 3.39 : Finite Automata for RE_1.

The NDFA for $(RE_2)^*$ is shown below.

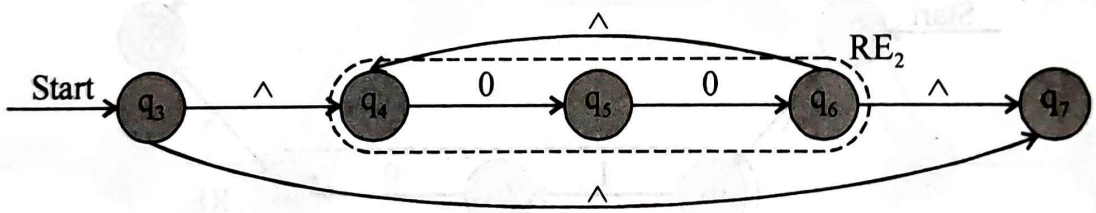


Fig. 3.40 : Finite Automata for $(RE_2)^*$

The resultant NDFA for the regular expression $RE = RE_1 (RE_2)^* = 0(0\ 0)^*$ is drawn below.

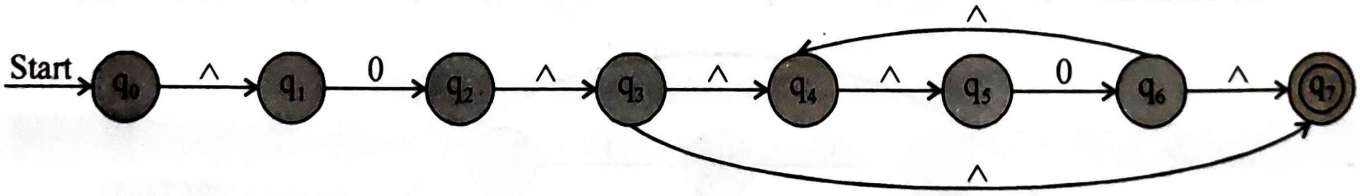


Fig. 3.41 : Finite Automata for $RE = 0(00)^*$.

Example 3.29 : Design the NDFA for the regular expression $RE = (01 + 2^*)1$ over the set of input alphabets $\Sigma = \{0, 1, 2\}$.

Solution : The regular expression for the $RE = (01 + 2^*)1$ can be divided as follows.

$$RE = (RE_1 + RE_2)\ RE_3 \text{ where,}$$

$$RE_1 = 01$$

$$RE_2 = 2^* \text{ and}$$

$$RE_3 = 1$$

Now, we draw the NDFA for RE_1, RE_2 and RE_3.

The NDFA for $RE_1 = 01$:

Fig. 3.42 : Finite Automata for RE_1.

The NDFA for $RE_2 = 2^*$:

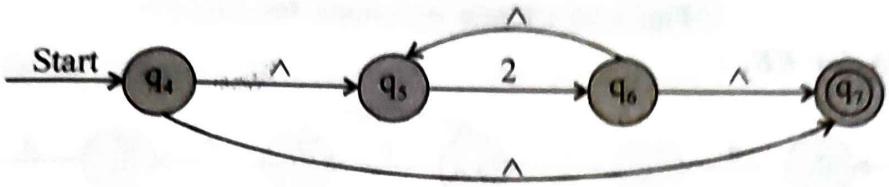


Fig. 3.43 : Finite Automata for RE_2.

The NDFA for $RE_3 = 1$:

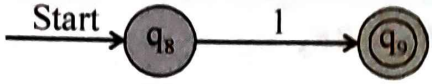


Fig. 3.44 : Finite Automata for RE_3.

Now, we draw the NDFA for resultant regular expression $RE = (RE_1 + RE_2)\ RE_3$

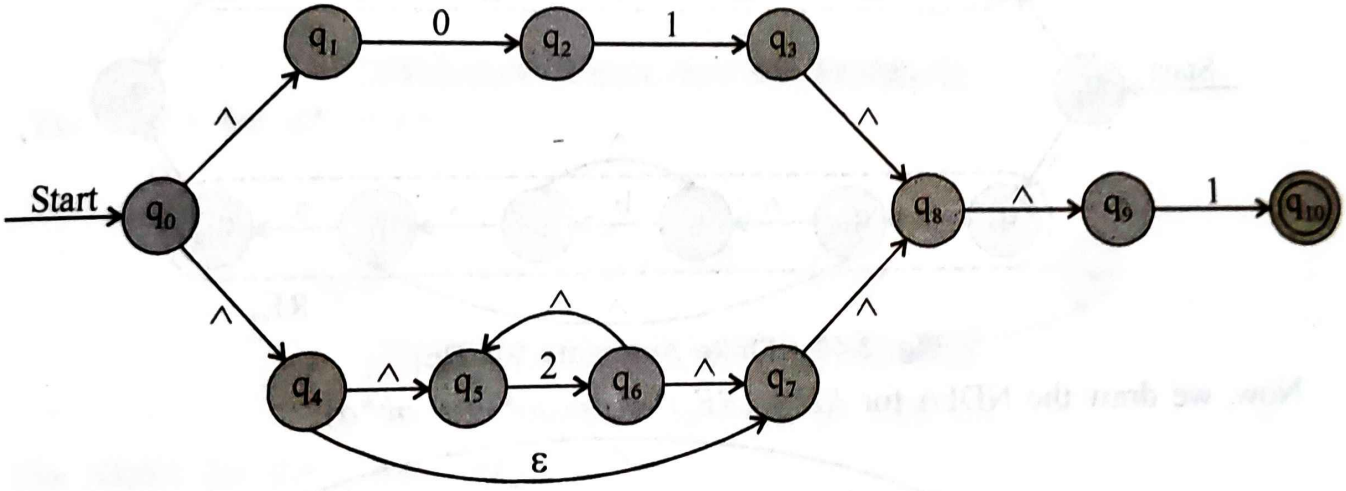


Fig. 3.45 : Finite Automata for $RE = (01 + 2^*)1$.

Example 3.30 : Design the NDFA for the regular expression $RE = (a(aa)^*b + ab^*a)^*$.

Solution : Given : $RE = (a(aa^*)b + ab^*a)^*$

The regular expression $RE = (a(aa^*)b + ab^*a)^*$ can be divided as follows :

$RE = (RE_1)^*$ where

$RE_1 = (a(aa^*)b + ab^*a)$.

The regular expression RE_1 can further be divided as follows :

$RE_1 = RE_2 + RE_3$ where,

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

$RE_3 = ab^*a$

Now, we draw the NDFA for RE_2 and RE_3.

NDFA for $RE_2 = a(aa^*)b$:

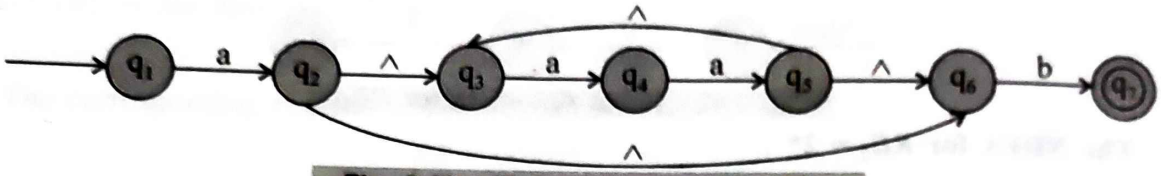


Fig. 3.46 : Finite Automata for RE_2.

The NDFA for RE_3 :

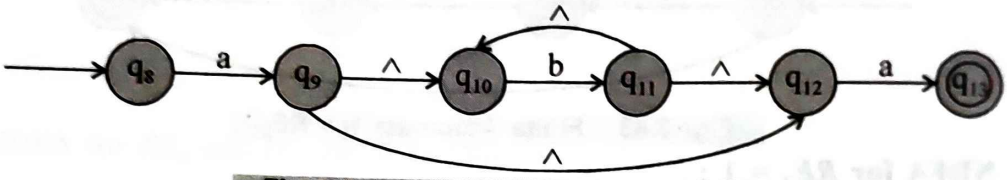


Fig. 3.47 : Finite Automata for RE_3.

The NDFA for $RE_1 = RE_2 + RE_3$

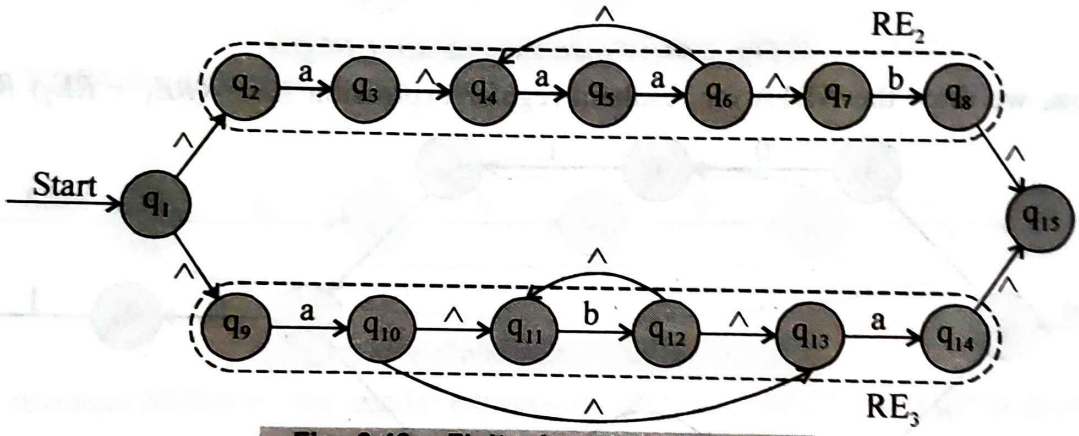


Fig. 3.48 : Finite Automata for RE_1.

Now, we draw the NDFA for $RE = (RE_1)^* = (a(aa^*)b + ab^*a)^*$

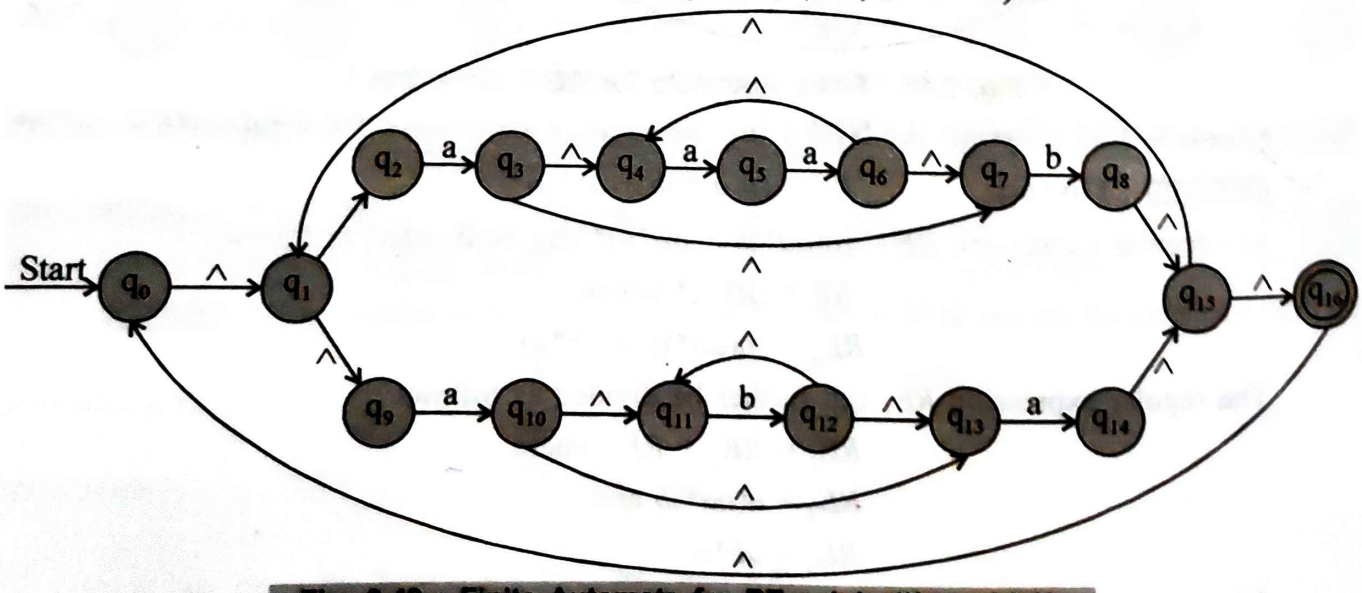


Fig. 3.49 : Finite Automata for RE = (a(aa*)b + ab*a)*.

Example 3.31 : Design the NDFA for the strings which do not contain the substring ab over the set of input alphebats $\Sigma = \{a, b\}$.

Solution : First, we find the regular expression for the strings which do not contain substring *ab* over $\Sigma = \{a, b\}$, i.e., we can't have *b* followed by *a*.

The regular expression for this set is :

$$RE = b^*a^*$$

We can divide the regular expression $RE = b^*a^*$ as follows :

$$RE = RE_1 \cdot RE_2 \text{ where,}$$

$$RE_1 = b^*$$

$$RE_2 = a^*$$

The NDFA for $RE_1 = b^*$:

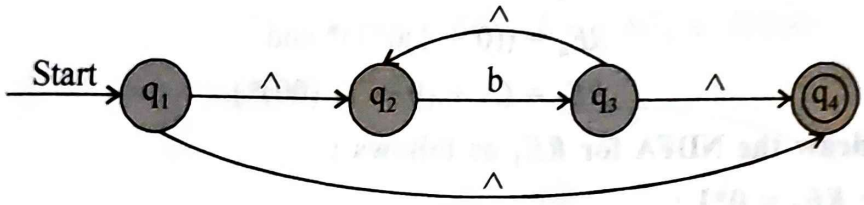


Fig. 3.50 : Finite Automata for RE_1.

The NDFA for $RE_2 = a^*$:

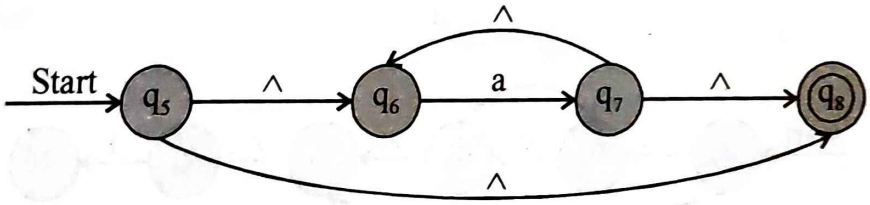


Fig. 3.51 : Finite Automata for RE_2.

The NDFA for $RE = RE_1 \cdot RE_2$:

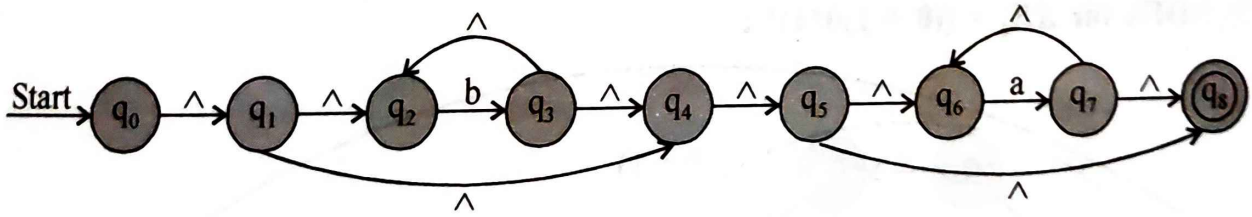


Fig. 3.52 : Finite Automata for $RE = b^*a^*$.

Reducing the above NDFA, we get,

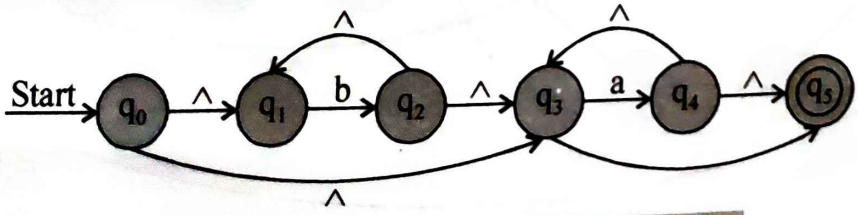


Fig. 3.53 : Finite Automata for $RE = b^*a^*$.

Example 3.32 : Design the NDFA for the regular expression
$0^*1\ ((0 + 1)0^*\ 1)^*\ (\wedge + (0 + 1)\ (00)^*) + 0(00)^*$

Solution : Given :

$$RE = 0^*1\ ((0 + 1)0^*1)^*\ (\wedge + (0 + 1)\ (00)^*) + 0(00^*).$$

This regular expression RE can be divided into small expressions as follows :

$$RE = RE_1 + RE_2 \text{ where,}$$

$$RE_1 = 0^*\ 1\ ((0 + 1)0^*\ 1)^*\ (\wedge + (0 + 1)\ (00)^*)$$

and $$RE_2 = 0(00)^*$$

Now, we can divide RE_1 as follows :

$$RE_1 = RE_3 \cdot RE_4 \cdot RE_5 \text{ where}$$

$$RE_3 = 0^*\ 1$$

$$RE_4 = ((0 + 1)0^*1)^* \text{ and}$$

$$RE_5 = (\wedge + (0 + 1)\ (00)^*)$$

Now, we draw the NDFA for RE_1 as follows :

NDFA for $RE_3 = 0^*1$:

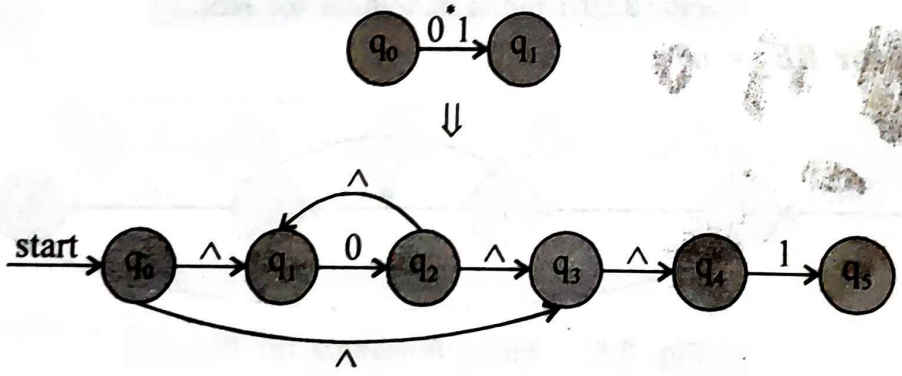


Fig. 3.54 : Finite Automata for RE_3.

NDFA for $RE_4 = ((0 + 1)0^*1)^*$:

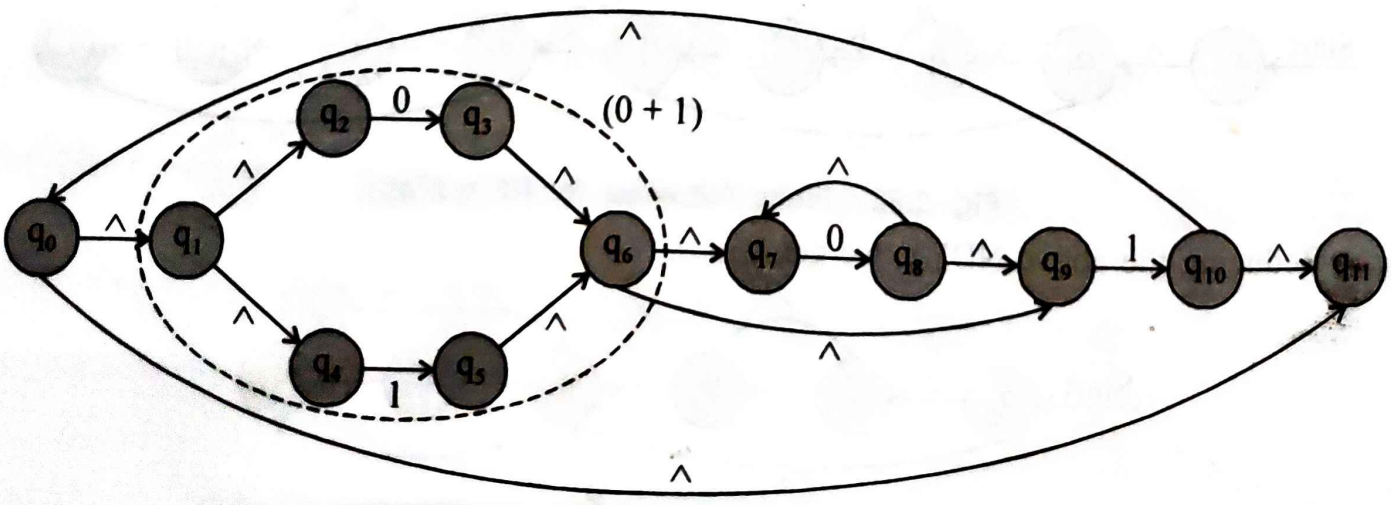


Fig. 3.55 : Finite Automata for RE_4.

NDFA for $RE_5 = (\wedge + (0 + 1)\ (00)^*)$:

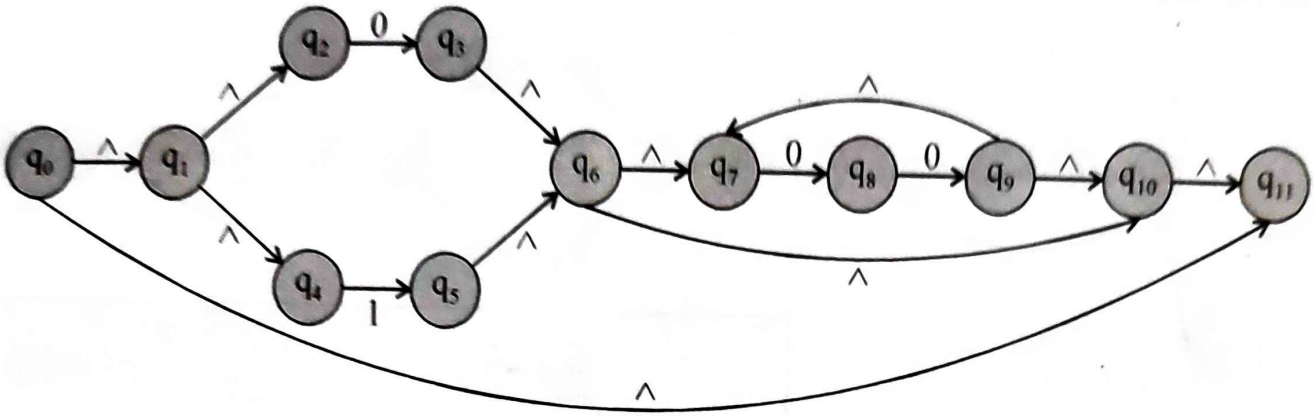


Fig. 3.56 : Finite Automata for RE_5.

Now, we combine the NDFA for $RE_1 = RE_3 \cdot RE_4 \cdot RE_5$ as follows

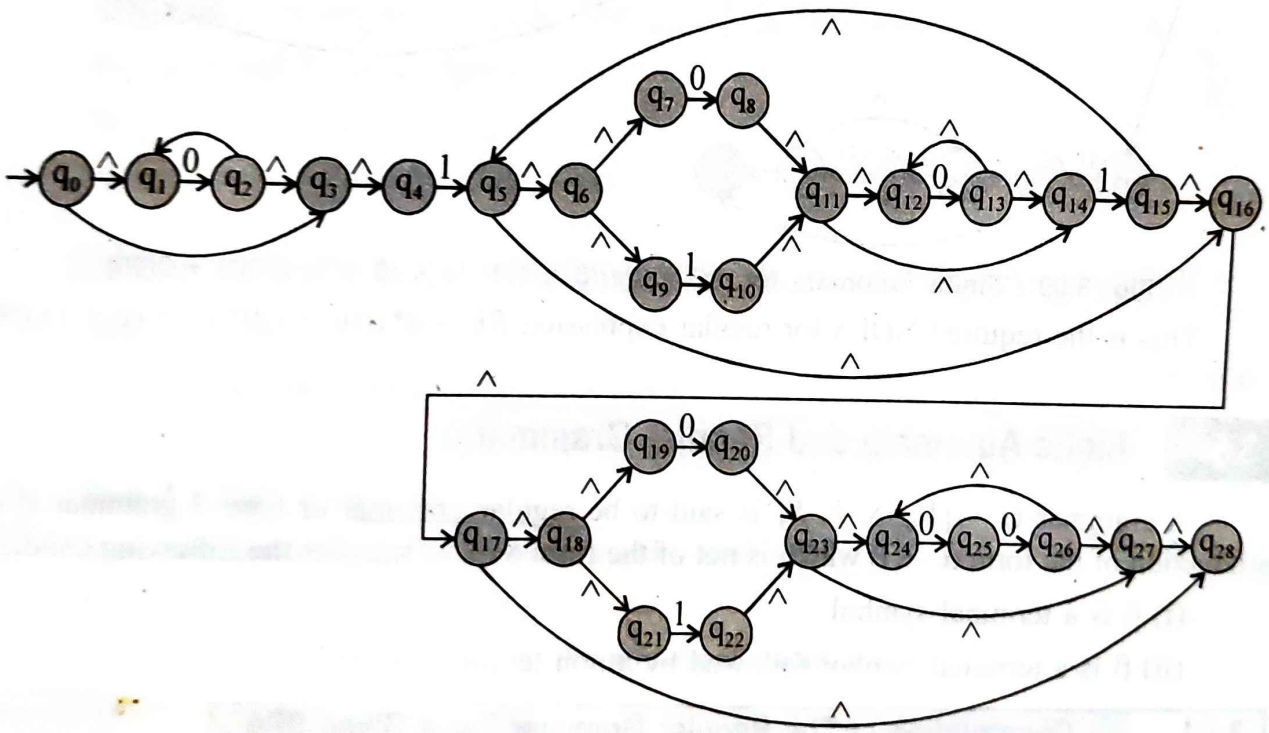


Fig. 3.57 : Finite Automata for RE_1.

Next, construct the NDFA for regular expression $RE_2 = 0(00)^*$:

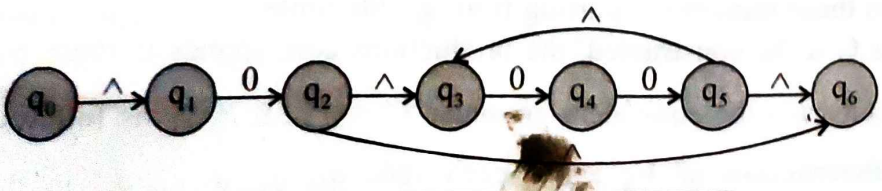


Fig. 3.58 : Finite Automata for RE_2.

Combining the regular expression RE_1 and RE_2, we get the regular expression $RE = RE_1 + RE_2$ as follows :

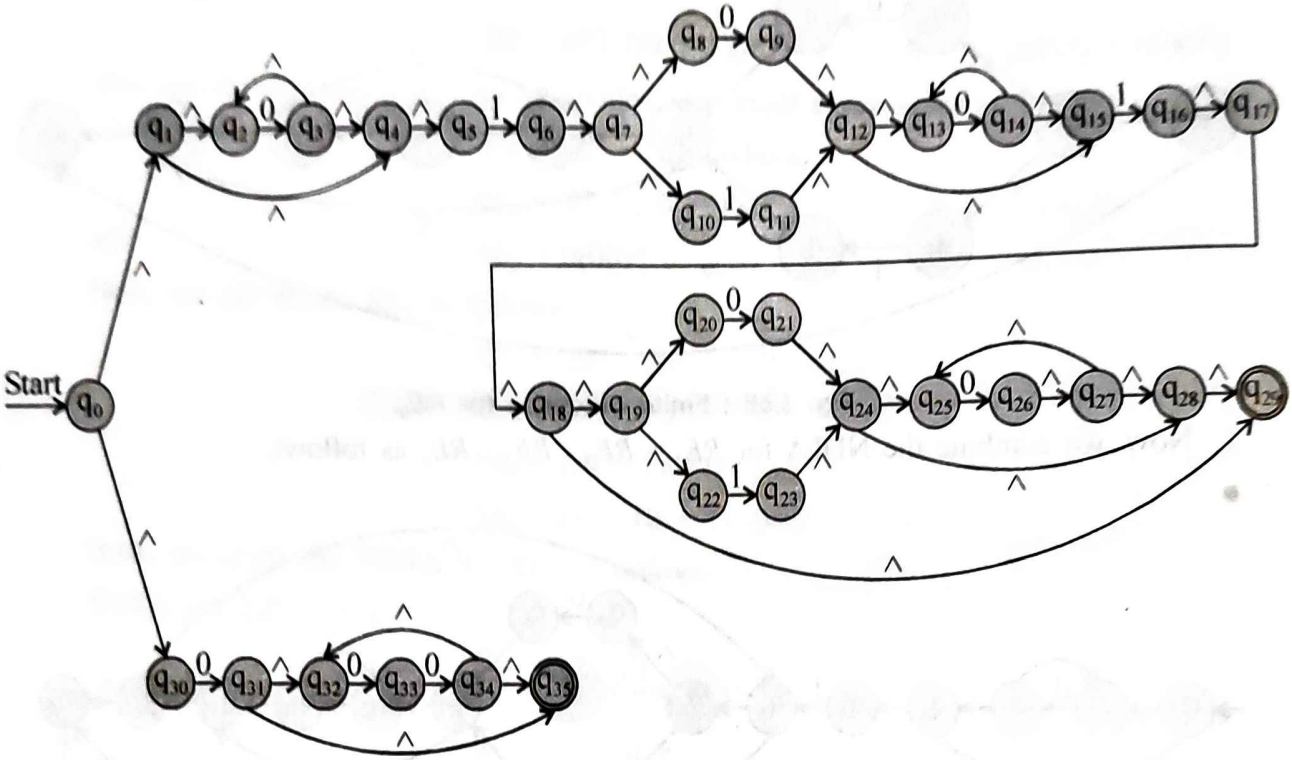


Fig. 3.59 : Finite Automata for RE = 0*1((0 + 1)0* (a + (0 + 1) (00)*) + 0(00)*.

This is the required NDFA for regular expression RE = 0*1((0 + 1)0* (a + (0 + 1) (00)*) + 0(00)*.

3.5 Finite Automata and Regular Grammar

A grammar $G = \{V_N, S, P, S\}$ is said to be regular grammar or type–3 grammar if each production of the form $\alpha \rightarrow \beta$ which is not of the form $S \rightarrow \wedge$, satisfies the following condition :

(i) β is a terminal symbol

(ii) β is a terminal symbol followed by a non terminal symbol.

3.5.1 Construction of The Regular Grammar For a Given DFA

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. If w is the string which is accepted by the given determinstic finite automata M. Then, this string will be obtained by concatenating the labels corrosponding to these transitions starting from q_0 (the initial state) to its q_f (the final state), So, for the grammar G to be constructed, the productions corrosponds to these transitions.

We can construct the equivalent grammar $G = \{V_N, \Sigma, P, S\}$ as **follows :**

1. **Construction of V_N :** For each state q_0, q_1, q_2 .... q_n, in the DFA, we take A_0, A_1, A_2, ... A_n non-terminal sybmols which form the elements of V_N.

2. **Construction of Σ :** All distinct labels in the transition graph/DFA are considered in a set of input alphabet Σ.

3. **Start Symbol (S) :** State corrosponding to the q_0 i.e., start state in transition graph is considered as the start symbol of the grammar (Here, it is A_0).

4. Construction of P **(Set of Productions)** :

(i) The production $A_i \rightarrow aA_j$ is added in P if $\delta(q_i, a) = q_j$ and $q_j \notin F$.

(ii) $A_i \rightarrow aAj$ and $A_i \rightarrow a$ are added in P if $\delta(q_i, a) = q_j$ and $q_j \in F$.

Example 3.33 : Construct the regular grammar for the deterministic finite automata shown in Fig. 3.60.

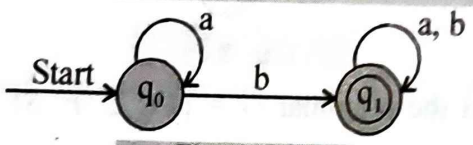


Fig. 3.60

Solution : We construct the grammar $G = \{V_N, \Sigma, P, S\}$ where, V_N, Σ, P and S are as follows

Step 1 : Construction of V_N.

As each state in the DFA corrosponds to the symbol in V_N and renaming these states as A_0, A_1 A_2 .... A_n, we get A_0 and A_1 since there are **two states** q_0 and q_1

$$\therefore \quad V_N = \{A_0, A_1\}$$

Step 2 : Construction of Σ.

As there are **two** distinct symbols **a** and **b**, which constitute the Σ

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

Step 3 : Start symbol (S) :

State corrosponding to the symbol q_0, i.e., start state = A_0

Step 4 : Set of productions (P) :

(a) Productions corrosponding to the **non-final states**, i.e.,

$$A_0 \rightarrow aA_0$$

(b) Production corrosponding to the final states

$$A_0 \rightarrow bA_1, \quad A_0 \rightarrow b$$

$$A_1 \rightarrow aA_1, \quad A_1 \rightarrow a$$

$$A_1 \rightarrow bA_1, \quad A_1 \rightarrow b$$

∴ The required grammar $G = (\{A_0, A_1\}, \{a, b\}, P, A_0)$

where P is given by set of the production rules derived in step 4.

Example 3.34 : Construct the regular grammar for the determinstic finite automata shown below in Fig. 3.61.

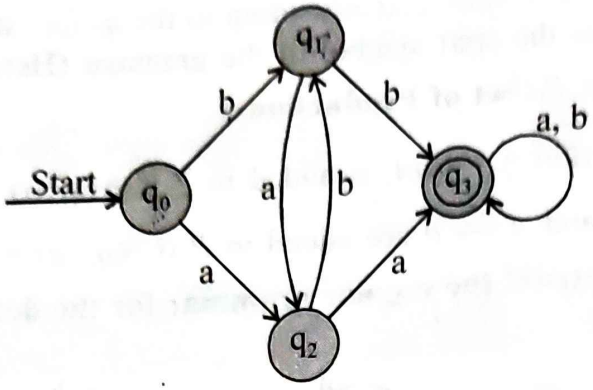


Fig. 3.61

Solution : We construct the grammar $G = \{V_N, \Sigma, P, S\}$ where,

(i) $V_N = \{A_0, A_1, A_2, A_3\}$

A_0, A_1, A_2 and A_3 corrosponding to the states q_0, q_1, q_2 and q_3

(ii) $\Sigma = \{a, b\}$

Set of non-terminals used in the transition graph.

(iii) $S = \{A_0\}$

Start state corrosponding to the state q_0 and

(iv) P (set of productions) are given as follows :

(a) Productions corrosponding to the **non-final states**

$$A_0 \rightarrow bA_1$$

$$A_0 \rightarrow aA_2$$

$$A_1 \rightarrow aA_2 \text{ and}$$

$$A_2 \rightarrow bA_1$$

(b) Productions corrosponding to the **final state.**

$$A_1 \rightarrow bA_3, \quad A_1 \rightarrow b$$

$$A_2 \rightarrow aA_3, \quad A_2 \rightarrow a$$

$$A_3 \rightarrow aA_3, \quad A_3 \rightarrow a$$

$$A_3 \rightarrow bA_3, \quad A_3 \rightarrow b$$

This is the required grammar.

Example 3.35 : Construct the regular grammar for the transition graph shown below in Fig. 3.62.

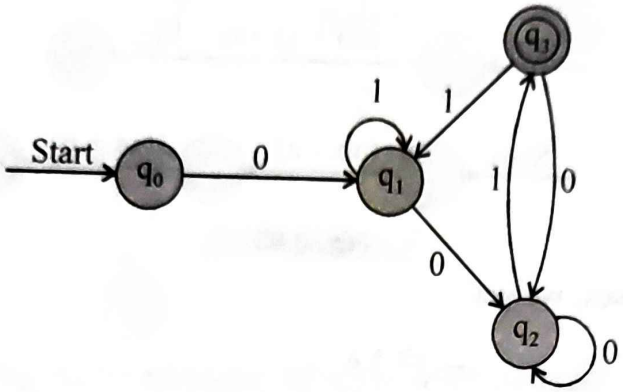


Fig. 3.62

Solution : The equivalent regular grammar $G = \{V_N, \Sigma, P, S\}$ can be formed as follows:

(i) $V_N = \{A_0, A_1, A_2, A_3\}$ where,

A_0, A_1, A_2 and A_3 non terminals are corrosponding to the states q_0, q_1, q_2 and q_3.

(ii) $\Sigma = \{0, 1\}$

(iii) $S = \{A_0\}$

(iv) P : Set of production rules are given as follows :

(a) Production corrosponding to the **non-final states**

$$A_0 \rightarrow 0A_1$$

$$A_1 \rightarrow 1A_1$$

$$A_1 \rightarrow 0A_2$$

$$A_2 \rightarrow 0A_2$$

$$A_3 \rightarrow 1A_1$$

$$A_3 \rightarrow 0A_2$$

(b) Productions corrosponding to the **final state**

$$A_2 \rightarrow 1A_3, \quad A_2 \rightarrow 1$$

This is the required grammar.

Example 3.36 : Construct the regular grammar $G = \{V_N, \Sigma, P, S\}$ for the regular expression

$$(ab + a)^* (aa + b)$$

Solution : First, convert the regular expression into NDFA,

Step 1 : $RE = (ab + a)^* (aa + b)$

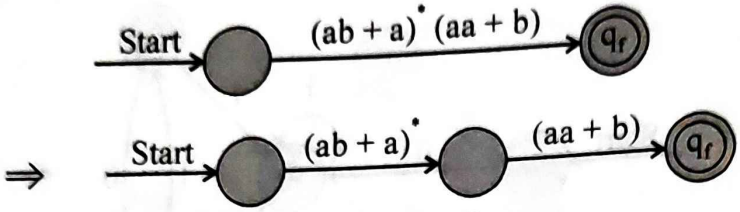


Fig. 3.63

By inserting ∧-moves, we get,

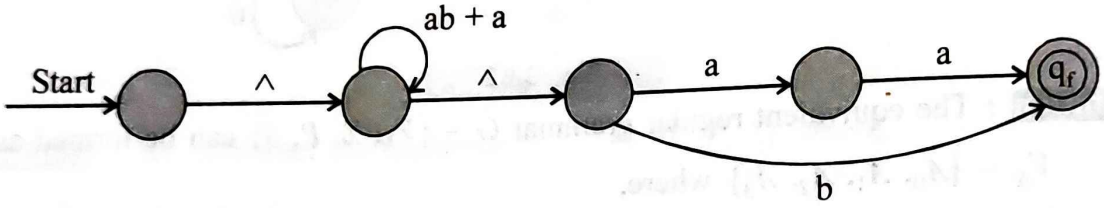


Fig. 3.64

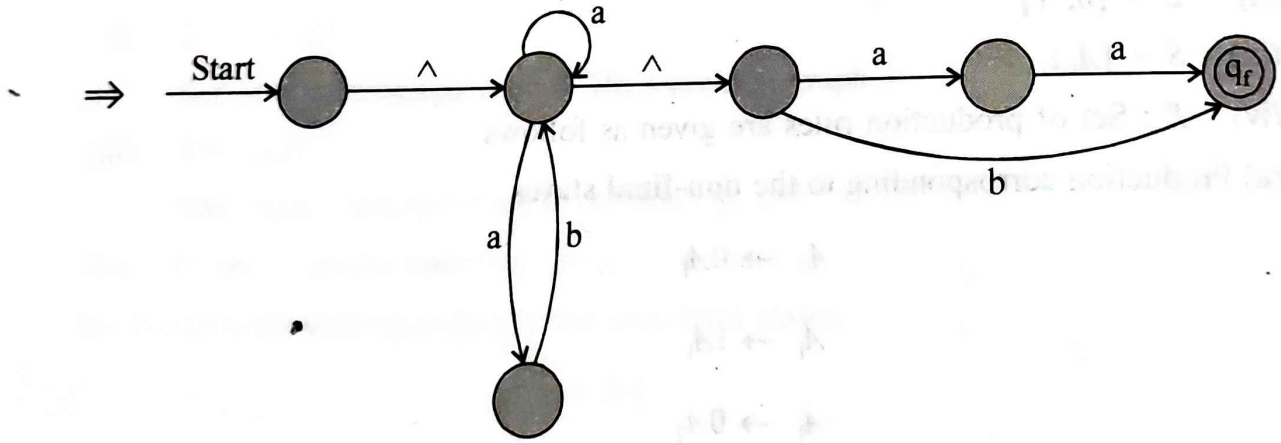


Fig. 3.65

Eliminating ∧-moves from the transition system, we get,

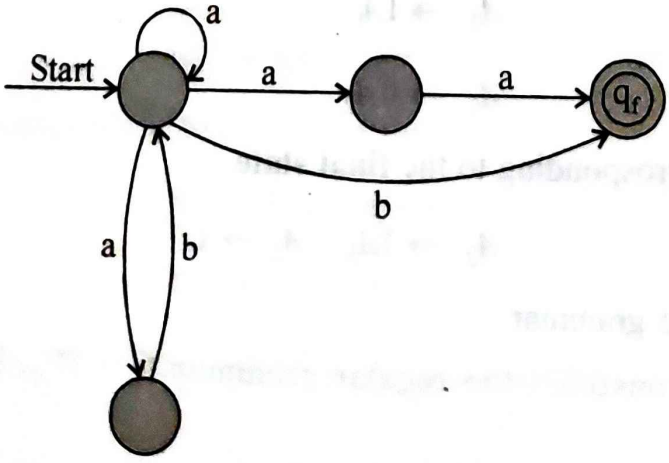


Fig. 3.66

Naming various unnamed states, we get,

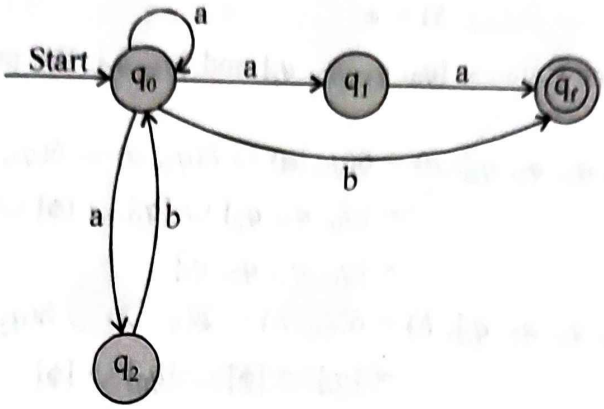


Fig. 3.67 : NDFA for *RE* = (*ab* + *a*)* (*aa* + *b*).

Step 2 : Construct the transition table :

State \ Input	a	b
→ q_0	$[q_0, q_1, q_2]$	q_f
q_1	q_f	ϕ
q_2	ϕ	q_0
(q_f)	ϕ	ϕ

Table 3.1

Step 3 : Now, we convert this NDFA to the DFA.

(i) Find reachable states :

We start with initial state q_0 and apply input symbols a and b as follows :

$$\delta(q_0, a) = [q_0, q_1, q_2] \quad \ldots(1)$$

$$\delta(q_0, b) = [q_f] \quad \ldots(2)$$

we get **two** new states, namely, $[q_0, q_1, q_2]$ and $[q_f]$.

We proceed for these states as follows :

$$\begin{aligned}\delta([q_0, q_1, q_2], a) &= \delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a) \\ &= [q_0, q_1, q_2] \cup [q_f] \cup [\phi] \\ &= [q_0, q_1, q_2, q_f] \quad \ldots(3)\end{aligned}$$

$$\begin{aligned}\delta([q_0, q_1, q_2], b) &= \delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b) \\ &= [q_f] \cup [\phi] \cup [q_0] \\ &= [q_0, q_f] \quad \ldots(4)\end{aligned}$$

$$\delta([q_f], a) = \phi \quad \ldots(5)$$

$$\delta([q_f], b) = \phi \quad \ldots(6)$$

Now, we get **two** new states, $[q_0, q_1, q_2, q_f]$ and $[q_0, q_f]$. We proceed for these states as follows :

$$\begin{aligned}\delta([q_0, q_1, q_2, q_f], a) &= \delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a) \cup \delta(q_f, a) \\ &= [q_0, q_1, q_2] \cup [q_f] \cup [\phi] \cup [\phi] \\ &= [q_0, q_1, q_2, q_f] \quad \ldots(7)\end{aligned}$$

$$\begin{aligned}\delta([q_0, q_1, q_2, q_f], b) &= \delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b) \cup \delta(q_f, b) \\ &= [q_f] \cup [\phi] \cup [q_0] \cup [\phi] \\ &= [q_0, q_f] \quad \ldots(8)\end{aligned}$$

$$\begin{aligned}\delta([q_0, q_f], a) &= \delta(q_0, a) \cup \delta(q_1, a) \\ &= [q_0, q_1, q_2] \cup [\phi] \\ &= [q_0, q_1, q_2] \quad \ldots(9)\end{aligned}$$

$$\begin{aligned}\delta([q_0, q_f], b) &= \delta(q_0, b) \cup \delta(q_f, b) \\ &= [q_f] \cup [\phi] \\ &= [q_f] \quad \ldots(10)\end{aligned}$$

Now, we don't get any new state.

The set of reachable states :

$[q_0]$, $[q_0, q_1, q_2]$, $[q_0, q_1, q_2, q_f]$, $[q_0, q_f]$ and $[q_f]$

(ii) Start state of the DFA = $[q_0]$

(As q_0 is the start state of the NDFA.)

(iii) Final states of the DFA = $[q_f]$, $[q_0, q_f]$ and $[q_0, q_1, q_2, q_f]$

(As these states contain final state of NDFA.)

Renaming the states obtained in step 2 as follows, we get,

$$[q_0] = A_0$$

$$[q_0, q_1, q_2] = A_1$$

$$[q_0, q_2, q_3, q_f] = A_2$$

$$[q_0, q_f] = A_3 \text{ and}$$

$$[q_f] = A_4$$

The DFA after renaming the states is shown in Fig. 3.68.

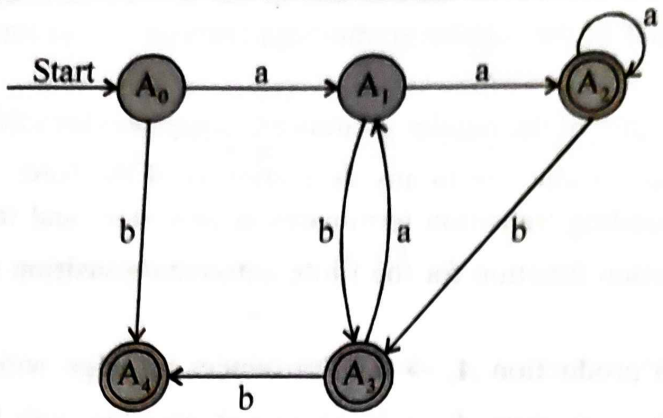


Fig. 3.68

Step 4 : Now, we construct the regular grammar from the DFA obtained in step 3.

$$V_N = \{A_0, A_1, A_2, A_3, A_4\}$$

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

S : Starting symbol and

P : Set of production rules are given by

(i) Production rules corrosponding to the **non-final states**.

$$A_0 \rightarrow aA_1$$

$$A_3 \rightarrow aA_1$$

(ii) Productions corrosponding to the **final states**.

$$A_0 \rightarrow bA_4, \quad A_0 \rightarrow b$$

$$A_1 \rightarrow aA_2, \quad A_1 \rightarrow a$$

$$A_1 \rightarrow bA_3, \quad A_1 \rightarrow b$$

$$A_2 \rightarrow aA_2, \quad A_2 \rightarrow a$$

$$A_2 \rightarrow bA_3, \quad A_2 \rightarrow b$$

$$A_3 \rightarrow bA_4, \quad A_3 \rightarrow b$$

$A_4 \rightarrow \wedge$ ($\because$ It is final state and have no out going edge).

3.5.2 Construction of Finite Automata From The Regular Grammar

Let $G = (V_N, \Sigma, P, S)$ be a regular grammar. We can construct the transition system M equivalent to the given regular grammar G by using following facts :

1. Non-terminals A_0, A_1, A_2, ... A_n of the grammar G corrosponds to each state q_0, q_1, ... q_n of the transition system.

2. Start symbol S of the regular grammar G corresponds to start state of the transition system.
3. Productions in P of the regular grammar G corresponds to the state transitions in M.
4. Since the last production in any derivation is of the form $A_i \rightarrow a$ for some a in Σ, the corresponding transition terminates at new state and this is unique final state.
5. δ, the transition function for the finite antomata/transition system M can be defined as follows :
   (i) Each production $A_i \rightarrow aA_j$ introduces an edge with label a from q_i to q_f
   (ii) Each production $A_i \rightarrow a$ introduces an edge with label a from q_i to q_f.

Example 3.37 : Construct the finite automata for the grammar

$$G = (\{S, A\}, \{a, b\}, P, S)$$

where, P is given by

$$S \rightarrow aS \mid bA \mid b$$
$$A \rightarrow aA \mid bS \mid a$$

Solution : Given

$$V_N : \{S, A\}$$
$$\Sigma : \{a, b\}$$
$$P : S \rightarrow aS \mid bA \mid b$$
$$A \rightarrow aA \mid bS \mid a$$
$$\text{Start state} : S$$

We take two states q_0 and q_1 corrosponding to the $V_N = \{S, A\}$ and one new state q_f as a accepting state/**final state** corrosponding to the production of the from $A_i \rightarrow a$.

Corelating each set of production rules to the various states of the finite automata; we get

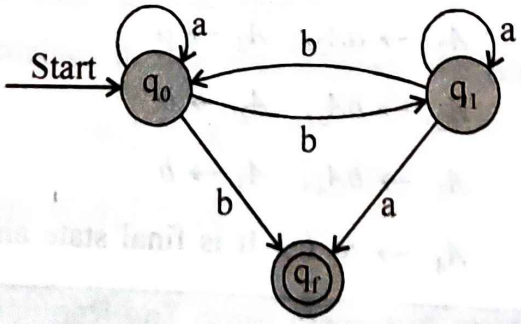


Fig. 3.69

An edge q_0 to q_1 with label b is added due to the production $S \rightarrow bA$.

Note : The edges from $q_0 \rightarrow q_f$ and $q_1 \rightarrow q_f$ are added due to the productions $S \rightarrow b$ and $A \rightarrow a$.

Example 3.38 : Construct the finite automata equivalent to the grammar shown below.

$$S \rightarrow aS \mid bS \mid aA$$

$$A \to bB$$
$$B \to aC$$
$$C \to A$$

Solution : Given :

$$V_N = \{S, A, B, C\}$$
$$\Sigma = \{a, b\}$$
S : Start state and
$$P : S \to aS \mid bS \mid aA$$
$$A \to bB$$
$$B \to aC$$
$$C \to A$$

We take four states q_0, q_1, q_2 and q_3 corrosponding to the $V_N = \{S, A, B, C\}$

Corrolating each production rule to the various states of the finite automata, we get,

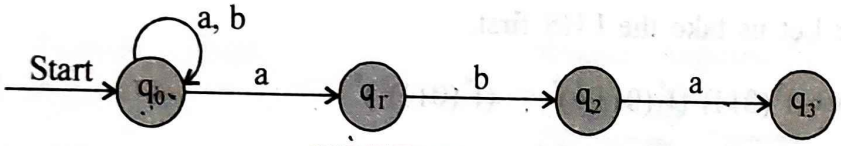


Fig. 3.70

The finite automata shown in Fig. 3.70 is the NDFA. (Readers are advised to convert it into the DFA).

3.6 Equivalence of Two Regular Expression

Let us suppose **P** and **Q** are the two regular expressions. These two regular expression are equivalent if and only if they represent the **same set of languages** and the **corrosponding finite automata are equivalent**.

To prove equivalence of two regular expressions, we perform either of the following tests :

1. In the two regular expressions P and Q, we find the string which is in one set, but not in other set, then the two regular expressions are not equivalent, otherwise, they are equivalent.
2. We use the **identities** to prove equivalence of two regular expressions P and Q.
3. We construct the finite automata $\mathbf{M_1}$ and $\mathbf{M_2}$ corrosponding to **two regular expressions** P and Q and prove the equivalence of finite automata and hence, the regular expressions.

Note : The method to be chosen to prove equivalence of two regular expressions depends on the problem.

Example 3.39 : Prove that

$$(1 + 00^*1) + (1 + 00^*1)\,(0 + 10^*1)^*\,(0 + 10^*1) = 0^*1(0 + 10^*1)^*.$$

Solution : Taking LHS

$$(1 + 00^*1) + (1 + 00^*\,1)\,(0 + 10^*1)^*\,(0 + 10^*\,1)$$

Taking $(1 + 00^*\,1)$ common, we get,

$$= (1 + 00^*1)(\wedge + (0 + 10^*1)^*(0 + 10^*1))$$

$$= (1 + 00^*1)(0 + 10^*1)^* \qquad [\because \wedge + RR^* = RR^* + \wedge = R^*]\ \text{(Identity } I_9)$$

$$= (\wedge + 00^*)1(0 + 10^*1)^* \qquad \text{[Taking 1 common from first term]}$$

$$= 0^*1(1 + 10^*1)^* \qquad [\because \wedge + 00^* = 0^*]\ \text{(Identity } I_9)$$

= RHS Hence, proved.

The two regular expressions are equivalent

Example 3.40 : Prove that $\wedge + 1^*(011)^*(1^*(011)^*)^* = (1 + 011)^*$

Solution : Let us take the LHS first,

$$\wedge + \underbrace{1^*(011)^*}_{R}\underbrace{(1^*(011)^*)^*}_{R^*} = (1^*(011)^*)^* \qquad \text{[using identity } I_9]$$

$$= (1 + 011)^* \qquad \text{[using identity } I_{11}]$$

= RHS Hence, proved.

∴ The two regular expressions are equivalent.

3.7 Closure Properties of Regular Expression

1. If the two languages L_1 and L_2 are regular, then the **union** of these two regular languages L_1 and L_2, i.e., $L_1 \cup L_2$ is also regular.
2. If the two languages L_1 and L_2 are regular, then the **intersection** of these two regular languages L_1 and L_2, i.e., $L_1 \cap L_2$ is also regular.
3. If the language L is regular then the **complement** of language L, i.e., $\overline{L}$ is also regular.
4. If the two languages L_1 and L_2 are regular then the **difference** of these two languages, i.e., $L_1 - L_2$ is also regular.
5. If the language $L(w)$ (Language L over string w) is regular then the reversal of language $L(w)$, i.e., $L(w^R)$ is also regular.
6. If the language $L(R)$ (Language L over regular expression R) is regular then the **closure** of language $L(R)$, i.e., $L(R^*)$ is also regular.

7. If the two languages L_1 and L_2 are regular than the **concatenation** of these two languages, i.e., L_1L_2 is also regular.
8. If the language L is regular over the alphabets Σ, and h is a **homomorphism** on Σ, then $h(L)$ is also regular.

   Note : A homomorphism is a function on strings that works by substituting a particular string for each symbol.

   Example : Suppose the string is 0100101 and the function is defined as follows: $h(0)$ = ab and $h(1) = c$ then string will become abcababcabc.

   Note : (0 is substituted by ab and 1 is substituted by c).
9. If h is a homomorphism from alphabet Σ to alphabet T, and L is a regular language over T, then $h^{-1}(L)$ is also a regular.

   For example, suppose L is regular language over $(ab)^*$, i.e., $h(L) = (ab)^*$.

   Let h be the homomorphism defined by $h(a) = 10$ and $h(b) = 01$.

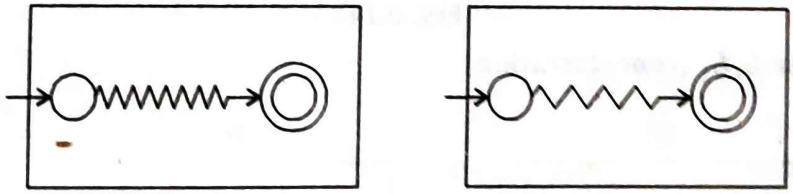
   Now, $h(L) = (1001)^*$ then, $h^{-1}(L)$ is the regular language over $(ba)^*$

Description of Some Properties

Let

Regular language L_1 — $L(R_1) = L_1$ — NFA R_1

Regular language L_2 — $L(R_2) = L_2$ — NFA R_2



Single accepting state — Single accepting state

Fig. 3.71

(1) NDFA for $L_1 \cup L_2$ (UNION)

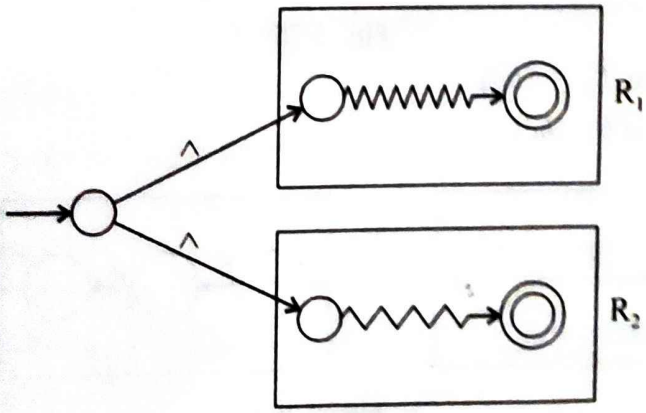


Fig. 3.72

Let $L_1 = \{a^n b\}$ and $L_2\{ba\}$

NDFA for $L_1 = \{a^n b\}$ NDFA for $L_2 = \{ba\}$

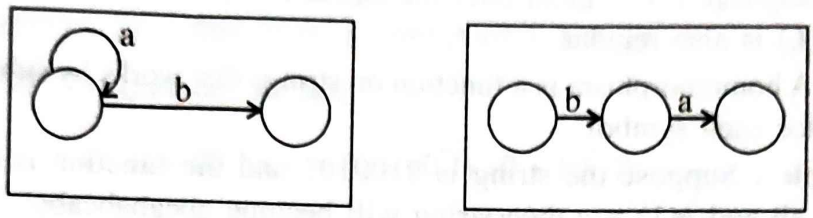


Fig. 3.73

Then, NDFA for $L_1 \cup L_2 = \{a^n b\} \cup \{bb\}$ is :

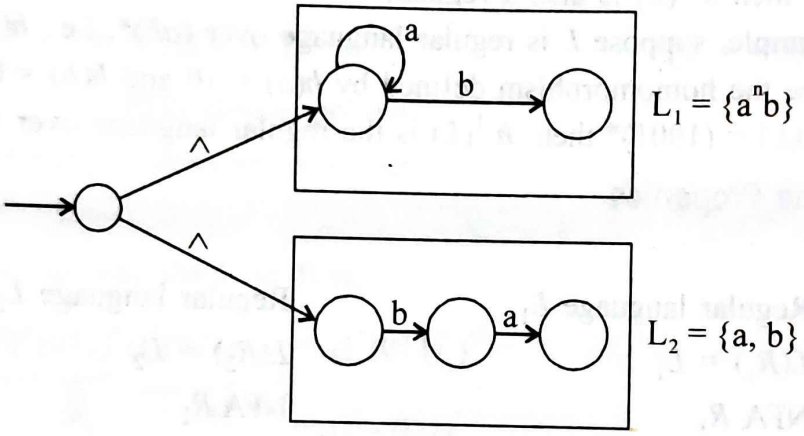


Fig. 3.74

(2) NDFA for L_1L_2 (concatatration)

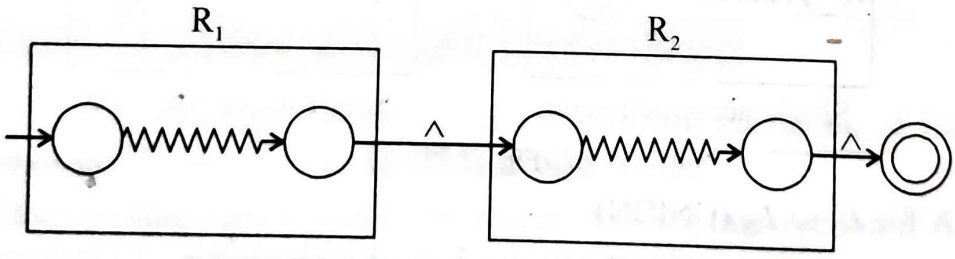


Fig. 3.75

Let $L_1 = \{a^n b\}$ and $L_2 = \{ba\}$

NFA for $L_1L_2 = \{a^n b\}\ \{bc\}$ is :

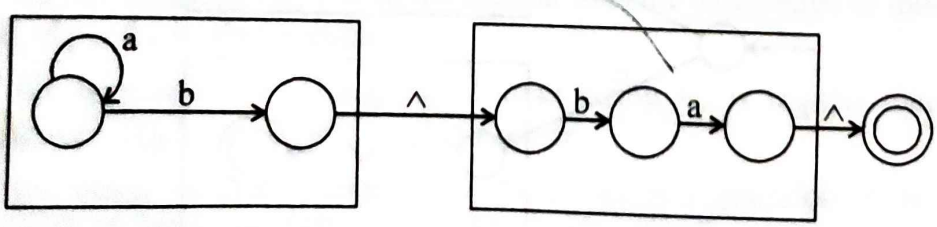


Fig. 3.76

3.8 Pumping Lemma for Regular Languages

If L is an infinite regular language, then there exists some positive integer m such that any string $w \in L$ whose length is m or greater can be decomposed into three parts, xyz, where

$|xy|$ is less than or equal to m,

$|y| > 0$,

$w_i = xy^iz$ is also in L for all $i = 0, 1, 2, 3, ...$

Note

- m is a (finite) number chosen so that strings of length m or greater *must* contain a cycle. Hence, m must be equal to or greater than the number of states in the *dfa*. Remember that we *don't know the dfa*, so we can't actually choose m; we just know that such an m must exist.
- Since string w has length greater than or equal to m, we can break it into two parts, xy and z, such that xy must contain a cycle. We don't know the dfa, so we don't know exactly where to make this break, but we know that $|xy|$ can be less than or equal to m.
- We let x be the part before the cycle, y be the cycle, and z the part after the cycle. (It is possible that x and z contain cycles, but we don't care about that.) Again, we don't know exactly where to make this break.
- Since y is the cycle we are interested in, we must have $|y| > 0$, otherwise it isn't a cycle.
- By repeating y an arbitrary number of times, xy^*z, we must get other strings in L.
- If, despite all the above uncertainties, we can show that the dfa has to accept some string that we know is not in the language, then we can conclude that the language is not regular.

To use this lemma, we need to show:

1. For *any* choice of m,
2. For some $w \in L$ that we get to choose (and we will choose one of length at least m),
3. For **any** way of decomposing w into xyz, so long as $|xy|$ is not greater than m and y is not $\in$,
4. We can choose an i such that xy^iz is not in L.

We can view this as a game wherein our opponent makes moves 1 and 3 (choosing m and choosing xyz) and we make moves 2 and 4 (choosing w and choosing i). Our goal is to show that we can **always** beat our opponent. If we can show this, we have proved that L is not regular.

Example 3.41 : Show that the set $L = \{VV^R : V \in \Sigma^*\}$ is not regular.

Solution : We use pumping lemma to show that the

$$L = \{VV^R : V \in \Sigma^*\} \text{ is not a regular.}$$

Assume for contradiction that L is a regular language.

$\because$ L is infinite, we can apply pumping lemma.

Let m be the integer in the Pumping lemma. We pick the string W such that $W \in L$ and length $|W| \geq m$

We pick $W = a^m b^m b^m a^m = xyz$

From the pumping lemma, it must be that length

$|xy| \leq m$, $|y| \geq 1$

$$\overbrace{\underbrace{a \ldots aa}_{x} \underbrace{\ldots a \ldots}_{y} \underbrace{a \ldots}_{m}}^{m} \overbrace{\underbrace{ab \ldots}_{m} \underbrace{bb \ldots}_{m} \underbrace{ba \ldots}_{m} \underbrace{a}_{m}}^{z}$$

Thus, $y = a^k, k \geq 1$

$xyz = a^m b^m b^m a^m$, $y = a^k$, $k \geq 1$

From the pumping lemma :

$xy^i z \in L \ \forall \ i \in 0, 1, 2$

Thus, $xy^2 z \in L$

$xyz = a^m b^m b^m a^m$ $\quad y = a^k, k \geq 1$

From the pumping lemma,

$xy^2 z \in L$

$$xy^2z = \overbrace{\underbrace{a \ldots aa}_{x} \underbrace{\ldots aa \ldots}_{y} \underbrace{a \ldots}_{y} \underbrace{a \ldots}_{m}}^{m} \overbrace{\underbrace{ab \ldots}_{m} \underbrace{bb \ldots}_{m} \underbrace{ba \ldots}_{m} \underbrace{a}_{m}}^{z} \in L$$

Thus, $a^{m+k} b^m b^m a^m \in L$; $k \geq 1$

But, $L = \{VV^R : V \in \Sigma^*\}$

$\Downarrow$

$a^{m+k} b^m b^m a^m \notin L$

(CONTRADICTION)

$\therefore$ Our assumption that L is a regular language is **not** true.

Conclusion : L is **not a regular language**.

Example 3.42 : Show that the set $L = \{a^n b^l c^{n+l} : n, l \geq 0\}$ is not regular.

Solution : Given : $L = \{a^n b^l c^{n+l} : n, l \geq 0\}$

We use pumping lemma to show that L is not regular.

Assume for contradiction that L is not regular

$\because$ L is infinite, we can apply pumping lemma.

Let m be the integer in the pumping lemma, Pick a string W such that $W \in L$ and length $|W| \geq m$.

We pick $W = a^m b^m c^{2m}$

We write, $a^m b^m c^{2m} = xyz$

From the pumping lemma

it must be that length $|xy| \leq m$; $|y| \geq 1$

$$xyz = \overbrace{\underbrace{a \ldots aa}_{m} \ldots \underbrace{aa}_{m}}^{x\ y} \ldots \overbrace{\underbrace{ab}_{m} \ldots \underbrace{bc}_{m} \ldots \underbrace{cc}_{2m} \ldots c}^{z}$$

Thus, $y = a^k,\ k \geq 1$

$$xyz = a^m b^m c^{2m},\quad y = a^k, k \geq 1$$

From the pumping lemma

$$xy^i \in L \text{ where } i = 0, 1, 2, \ldots$$

Thus, $xy^0 z = xz \in L$

$$xyz = a^m b^m c^{2m},\qquad y = a^k, k \geq 1$$

From the pumping lemma,

$$xz \in L$$

$$xz = \overbrace{\underbrace{a \ldots aa}_{m-k} \ldots \underbrace{ab}_{m} \ldots \underbrace{bc}_{m} \ldots \underbrace{cc}_{2m} \ldots c}^{x\ \ z} \in L$$

Thus, $a^{m-k} b^m c^{2m} \in L,\quad k \geq 1$

But $L = \{a^n b^l c^{n+l} : n, l \geq 0\}$

$$\Downarrow$$

$$a^{m-k} b^m c^{2m} \notin L$$

(Contradiction)

$\therefore$ Our assumption that L is a regular language is **not** true.

Conclusion : L is not regular.

Example 3.43 : Show that the set $L = \{a^{n!} : n \geq 0\}$ is not regular.

$$n! = 1 \cdot 2 \cdot 3 \ldots (n-1) \cdot n$$

Proof : Given : $L = \{a^{n!} : n \geq 0\}$

We use the pumping lemma.

Assume for contraction that L is a regular language.

Since L is infinite, we can apply the pumping lemma.

Let m be the integer in pumping lamma.

Pick a string W such that

$$W \in L$$

length $|W| \geq m$

we pick $W = a^{m!}$

We write $a^{m!} = xyz$

From the pumping lemma, it must be that length $|xy| \leq m$, $|y| \geq 1$

$$xyz = a^{m!} = \overbrace{\underbrace{a \ldots aa}_{x} \underbrace{\ldots aa}_{y} \ldots aa}^{m} \overbrace{\ldots aa \ldots a}^{m!-m}$$

Thus, $y = a^k, \quad 1 \leq k \leq m$

$$xyz = a^{m!} \; ; \quad y = a^k, 1 \leq k \leq m$$

From the pumping lemma

$xy^iz \in L$ where $i = 0, 1, 2 \ldots$

Thus, $xy^2z \in L$

$$xyz = a^{m!} \; ; \quad y = a^k, 1 \leq k \leq m$$

From the pumping lemma

$$xy^2z \in L$$

$$xy^2z = \overbrace{\underbrace{a \ldots aa}_{x} \underbrace{\ldots aa}_{y} \underbrace{\ldots aa}_{y} \ldots aa}^{m+k} \overbrace{\ldots aa \ldots a}^{m!-m} \in L$$

Thus, $a^{m+k} \in L \; ; \quad 1 \leq k \leq m$

Since, $L = \{a^{n!} : n \geq 0\}$

$$\Downarrow$$

There must exist p such that : $m! + k = p!$

However, $m! + k \leq m! + m$ For $m > 1$

$$\leq m! + m!$$

$$< m!m + m!$$

$$= m!(m + 1)$$

$$= (m + 1)!$$

$$\Downarrow$$

$$m! + k < (m + 1)!$$

$$\Downarrow$$

$$m! + k \neq p \text{ for any } p$$

$$a^{m!+k} \in L \qquad 1 \leq k \leq m$$

But, $L = \{a^{n!} : n \geq 0\}$

$$\Downarrow$$

$$a^{m!+k} \notin L$$

(CONTRADICTION)

$\therefore$ Our assumption that L is a regular language is not true.

Conclusion : L is not regular language.

Example 3.44 : Show that the set $L = \{a^n b^n : n \geq 0\}$.

Solution : Given $L = \{a^n b^n : n \geq 0\}$

Assume for contradiction that L is a regular language.

$\because$ L is infinite, we can apply pumping lemma.

Let us consider the integer m much that $W \in L$ and $|W| \geq m$

Now, we pick,

$$W = a^m b^m$$

We write,

$$a^m b^m = xyz$$

From the pumping lemma, it must be that :

length $|xy| \leq m, |y| \geq 1$

$$\therefore \quad a^m b^m = \underbrace{a \ldots aa \ldots a}_{x\ \ y}\ \underbrace{\ldots ab \ldots b}_{z}$$

$$y = a^k, \quad k \geq 1$$

We have,

$$xyz = a^m b^m, y = a^k, k \geq 1$$

From the pumping lemma :

$$xy^i z \in L, \quad \forall i = 0, 1, 2 \ldots$$

Thus, $xy^2 z \in L$

$$xy^2 z = xyyz = a^{m+k} b^m \in L$$

$$\therefore \quad a^{m+k} b^m \in L$$

But, $L = \{a^n b^n : n \geq 0\}$

$$\Downarrow$$

$$a^{m+k} b^m \in L$$

(CONTRADICTION)

$\therefore$ Our assumption that L is a regular language is not true.

Conclusion : L is not regular.

Example 3.45 : Prove that $L = \{0^i \mid i$ is a perfect square$\}$ is not a regular language.

Solution : Assume that L is regular and let m be the integer guaranteed by the pumping lemma. Now, consider the string $w = 0^{m\hat{}2}$. Clearly $w \in L$, so w can be written as $w = xyz$ with $|xy| \leq m$ and y $\neq \lambda$ (or $|y| > 0$). Consider what happens when $i = 2$. That is, look at xy^2z. Then, we have $m^2 = |w| < |xy^2z| \leq m^2 + m = m(m + 1) < (m + 1)^2$. That is, the length of the string xy^2z lies between two consecutive perfect squares. This means $xy^2z \notin L$ contradicting the assumption that L is regular.

Example 3.46 : Prove that $L = \{ww \mid w \in \{a, b\}^*\}$ is not regular.

Solution : Assume L is regular and let m be the integer from the pumping lemma. Choose $w = a^m b a^m b$. Clearly, $w \in L$ so by the pumping lemma, $w = xyz$ such that $|xy| \leq m$, $|y| > 0$ and $xy^iz \in L$ for all $i \geq 0$. Let $p = |y|$. Consider what happens when $i = 0$. The resulting string, $xz = a^{m-p} b a^m b$. Since $p \geq 1$, the number of a's in the two runs are not the same, and thus this string is not in L. Therefore L is not regular.

Example 3.47 : Prove that $L = \{a^n b^k : n > k \text{ and } n \geq 0\}$ is not regular.

Solution :

1. We don't know m, but assume there is one.
2. Choose a string $w = a^n b^k$ where $n > m$, so that any prefix of length m consists entirely of a's, and $k = n - 1$, so that there is just one more a than b.
3. We don't know the decomposition of w into xyz, but since $| xy | \leq m$, xy must consist entirely of a's. Moreover, y cannot be empty.
4. Choose $i = 0$. This has the effect of dropping $| y |$ a's out of the string, without affecting the number of b's. The resultant string has fewer a's than before, so it has either fewer a's than b's, or the same number of each. Either way, the string does not belong to L, so L is not regular.

Example 3.48 : Prove that $L = \{a^n : n$ is a prime number$\}$ is not regular.

Solution :

1. We don't know m, but assume there is one.
2. Choose a string $w = a^n$ where n is a prime number and $| xyz | = n > m + 1$. (This can always be done because there is no largest prime number.) Any prefix of w consists entirely of a's.
3. We don't know the decomposition of w into xyz, but since $| xy | \leq m$, it follows that $| z | > 1$. As usual, $| y | > 0$,
4. Since $| z | > 1$, $| xz | > 1$. Choose $i = | xz |$. Then $| xy^i z | = | xz | + | y | | xz | = (1 + | y |) | xz |$. Since $(1 + | y |)$ and $| xz |$ are each greater than 1, the product must be a composite number. Thus $| xy^i z |$ is a composite number.

3.9 Decision Algorithm for Regular Languages

What questions about regular languages can be answered and how long does it take to answer them? In answering such questions, we are free to convert the given description of the language(s) to any other representation of the language, though this conversion takes time.

The following problems are all decidable for regular languages.

1. Emptiness : Is a given language empty, i.e., does $L = \phi$?
   (a) Use structural induction on a regular expression for L : ϕ is empty; $\wedge$ and a $\in \Sigma$ are not empty; $E_1 + E_2$ is empty if and only if both E_1 and E_2 are empty; $E_1 E_2$ is empty if and only if either E_1 or E_2 is empty; E^* is not empty.
   (b) Use reachability in a DFA for L : L is empty if and only if there is no path from the initial state to an accepting state. This can be checked by a breadth-first traversal of the DFA from q_0.
2. Finiteness : Is the given language finite?

(a) Use structual induction on a regular expression for L. (More complex than finite ness).

(b) Let n be the pumping-lemma constant for L. Then L is finite if and only if it contains no string whose length is between n and $2n - 1$. This requires testing $0(|\Sigma|^{2n})$ strings for membership in L.

(If L has a string whose length is at least n, then, by the pumping lemma, L is infinite. If the shortest string w whose length is at least n is at least $2n$, then by the pumping lemma again, $w = xyz$, where $|xy| \leq n$ and $|y| \geq 1$, $xz \in L$. But $n \geq |xz| < |xyz| = |w|$, which is a contradiction.)

3. Membership does a given string belong to the given language, i.e., does $w \in L$? Convert the representation of L to a DFA M, if necessary. Then apply M to w, and see whether M accepts w, requiring time $O(|w|)$. But the time to convert a regular expression (or NDFA) to a DFA can be exponential in the size of the regular expression.

We can also simulate an NDFA directly, maintaining the current set of possible states, requiring time $O(|w| \times |Q|^2)$, which is what is done in practice.

4. Equality : Do two language descriptions define the same language, i.e., does $L_1 = L_2$? Clearly, $L_1 = L_2$ if and only if $L_1 \subseteq L_2$ and $L_2 \subseteq L_1$. But $L_1 \subseteq L_2$ if and only if $L_1 \cap L_2 = \phi$. We can compute the regular language $L_1 \cap L_2$ and we can decide whether or not it is empty. And similarly for $L_2 \subseteq L_1$. So the problem is decidable.

3.10 Myhill-Nerode Theorem

The Myhill-Nerode theorem is an important characterization of regular languages, and it also has many practical implications. We have learned regular languages, their properties and their usefulness for describing various systems. There are, however, languages that are not regular and therefore require devices other than finite automata to recognize them. In this section, we are going to study some of the methods for testing given languages for regularity and see some of the languages that are not regular. The main idea behind these test methods is that finite automata have only finite amount of memory in the form of states and that they can not distinguish infinitely many strings. For example to recognize the language $\{a^n b^n \mid n \text{ is a natural number}\}$, a finite automaton must remember how many a's it has read when it starts reading b's. Thus it must be in different states when it has read different number of a's and starts reading the first b. But any finite automaton has only finite number of states. Thus, there is no way for a finite automaton to remember how many a's it has read for all possible strings $a^n b^n$. That is the main limitation of finite automata. Since, a regular language must be recognized by a finite automaton, we can conclude that $\{a^n b^n \mid n \text{ is a natural number}\}$ is not regular. This is the basis of the regularity test methods Myhill-Nerode Theorem and Pumping Lemma.

Non-regularity test based on Myhill-Nerode's theorem

Indistinguishability of strings: Strings x and y in Σ^* are **indistinguishable with respect to a language** L if and only if for every string z in Σ^*, either xz and yz are both in L or they are both not in L.

For example, a and aa are indistinguishable with respect to the language a^n over alphabet {a}, where n is a positive integer, because aa^k and aaa^k are in the language a^n for any positive integer k. However, with respect to the language a^nb^n, a and aa are not indistinguishable (hence distinguishable), because ab is in the language a^nb^n while aab is not in the language.

Using this concept of indistinguishability, the following theorem by Myhill and Nerode gives a criterion for (non)regularity of a language. It is stated as:

A language L over alphabet Σ is non-regular if and only if there is an infinite subset of Σ, whose strings are pair wise distinguishable with respect to L.

Now, we give formal statement of the Myhill-Nerode Theorem:

Let $L \subseteq \Sigma^*$. Then the following statements are equivalent.

1. There is a DFA that accepts L. (L is regular language)
2. There is a right-invariant equivalence relation ~ of finite index such that L is the union of some of the equivalence class ~.
3. ~L is of finite index.

Note :

Equivalence Relation : An equivalence relation, ~, is a binary relation on a set if it is

→ Reflexive $(a \sim a)$

→ Symmetric (if $a \sim b$ then $b \sim a$)

→ Transitive (if $a \sim b$ and $b \sim c$ then $a \sim c$)

Equivalence Class : An equivalence relation ~ on a set S imposes a partition $S / \sim$ on S, such that elements e, $e' \in S$ are in the same part of $S / \sim$ if $e \sim e'$. The parts of $S / \sim$ are equivalence classes and $[e]$ is used to denote the unique equivalence class of $S / \sim$ to which e belongs.

The index of equivalence relation ~ is $| S / \sim |$

Right Invariant : The equivalence relation ~ on strings of symbols from some alphabet Σ is said to be right invariant if $\forall\ x, y \in \Sigma^*$ with $x \sim y$ and all $w \in \Sigma^*$, we have $xw \sim yw$.

This definition states that an equivalence relation has the right invariant property if two equivalent string (x and y) that are in the language still are equivalent if a third string (w) is appended to the right of both of them.

3.11 Applications of Regular Expression

1. Regular expressions in Unix

In the UNIX operating system various commands use an extended regular expressions language that provide shorthands for many common expressions. In this we can write character classes (A *character class* is a pattern that defines a set of characters and matches exactly one

character from that set.) to represent large set of characters. There are some rules for forming this character classes:

The dot symbol (.) is to represent 'any character'.

The regular expression a+b+c+...+z is represented by [abc...z]

Within a character class representation, - can be used to define a set of characters in terms of a range. For example, a-z defines the set of lower-case letters and A-Z defines the set of upper-case letters. The endpoints of a range may be specified in either order (i.e. both 0-9 and 9-0 define the set of digits).

If our expression involves operators such as minus then we can place it first or last to avoid confusion with the range specifier. i.e. [-.0-9]. The special characters in UNIX regular language can be represented as characters using \ symbol i.e. \ provides the usual escapes within character class brackets. Thus [[\]] matches either [or], because \ causes the first] in the character class representation to be taken as a normal character rather than the closing bracket of the representation.

Special notations

[: digit :]	same as [0-9]
[: alpha:]	same as [A-Za-z]
[: alnum :]	same as [A-Za-z0-9]

Operators

|	Used in place of +
?	0 or 1 of
R?	Means 0 or 1 occurrence of R
+	1 or more of
R+	means 1 or more occurrence of R
{n}	n copies of
R {3}	means RRR
^	Compliment of

If the first character after the opening bracket of a character class is ^, the set defined by the remainder of the class is complemented with respect to the computer's character set. Using this notation, the character class represented by '.' can be described as [^\n]. If ^ appears as any character of a class except the first, it is not considered to be an operator. Thus [^abc] matches any character except a, b, or c but [a^bc] or [abc^] matches a, b, c or ^.

When more than one expression can match the current character sequence, a choice is made as follows:

1. The longest match is preferred.
2. Among rules, which match the same number of characters, the rule given first is preferred.

2. Lexical analysis

Compilers – in a nutshell

Purpose: translate a program in some language (the *source language*) into a lower-level language (the *target language*).

Phases:

- Lexical Analysis:

  Converts a sequence of characters into words, or *tokens*
- Syntax Analysis:

  Converts a sequence of tokens into a *parse tree*
- Semantic Analysis:

  Manipulates parse tree to verify symbol and type information
- Intermediate Code Generation:

  Converts parse tree into a sequence of intermediate code instructions
- Optimization:

  Manipulates intermediate code to produce a more efficient program
- Final Code Generation:

  Translates intermediate code into final (machine/assembly) code

Overview of Lexical Analysis

- Convert character sequence into tokens, skip comments & whitespace
- Handle lexical errors
- Efficiency is crucial
- Tokens are specified as regular expressions, e.g., IDENTIFIER = [*a-z A-Z*] [*a-z A-Z* 0-9]*
- Lexical Analyzers are implemented by regular expressions.

There is a problem that more than one token may be recognized at once. Suppose the string **else** matches for regular expression as well as the expression for identifiers. This problem is resolved by giving priority to first expression listed.

Review Questions

1. Find regular expressions of the following sets :

   (i) The set of all strings over $\{a, b\}$ consisting exactly two b's or exactly three b's, not more.

   (ii) The set of all strings over $\{a, b\}$ in which the letter b is never tripled.