7 Finite Automata and Regular Expressions

'Enjoy the beauty in the development of the subject through expressions.'

Introduction

Regular expressions have an important role in computer science applications. For example, in applications involving text, if users are interested in searching for strings that satisfy certain patterns, then *regular expressions* certainly provide a powerful method for describing such patterns. Further, utilities like AWK and GREP in UNIX, text editors and modern programming languages such as PERL etc., provide mechanisms for the meaningful description of patterns using regular expressions. In chapter 3, regular expressions and languages were discussed in detail. In this chapter, equivalence of DFA and NFA with regular expressions is discussed. We briefly recall the definition of a *regular language* as the language represented by a regular expression. If r is a regular expression, then $L(r)$ denotes the language associated with r. This language is defined, formally, as follows:

- ϕ is a RE denoting the empty set.
- $\in$ is a RE denoting $\{\in\}$.
- For every $a \in \Sigma$, a is a RE denoting $\{a\}$.

7.1 Properties of Regular Sets or Regular Languages

7.1.1 *Closure Properties*

Theorem I: Regular Languages are closed under union, concatenation and kleene closure (closure properties)

Proof: Method-1

Given two regular languages – L_1 and L_2, if R_1 and R_2 are two REs, then by the definition of RE,

a. $R_1 + R_2$, denotes the union of two regular languages L_1 and $L_2 \Rightarrow L_1 \cup L_2$ is also regular.

b. $R_1 \cdot R_2$, denotes the union of two regular languages L_1 and $L_2 \Rightarrow L_1L_2$ is also regular.

c. R_1^* denotes the kleene closure of $L_1 \Rightarrow L_1^*$ is also regular.

EXAMPLE 7.1.1: (method 1)

(i) If R_1 and R_2 are REs with $R_1 = \{ab, c\}$ and $R_2 = \{d, ef\}$, then

$$\Rightarrow R_1 \cup R_2 = \{ab, c\} \cup \{d, ef\}$$
$$= \{ab, c, d, ef\}$$
$$\Rightarrow R_1R_2 = \{ab, c\} \cdot \{d, ef\}$$
$$= \{abd, cd, abef, cef\}.$$
$$\text{If } R = \{ab, c\}$$
$$\Rightarrow R^* = \{\in, c, abab, abc, cab, cc, \ldots\}.$$

(ii) If $r = a + b$, then the language of r is

$$L(r) = L(a + b)$$
$$= L(a) \cup L(b)$$
$$= \{a\} \cup \{b\}$$
$$\Rightarrow L(r) = \{a, b\} \text{ is also regular.}$$

(iii) If $r = ab^*a$, then the language of r is

$$L(r) = L(ab * a)$$
$$= L(a) \cdot L(b^*) \cdot L(a)$$
$$= \{a\} \cdot \{L(b)\}^*\{a\}$$
$$= \{a\} \cdot \{\in, b, bb, bbb, \ldots\} \cdot \{a\},$$
$$\Rightarrow L(r) = \{aa, aba, abba, \ldots\} \text{ is also regular.}$$

(iv) If $r = a^*$, then the language of r is

$$L(r) = L(a^*)$$
$$= (L(a))^*$$
$$\Rightarrow L(r) = \{\epsilon, a, aa, \ldots\} \text{ is also regular.}$$

Proof: Method-2

Assume that FA_1 and FA_2 are two finite automata's accepting languages L_1 and L_2 defined by regular expressions r_1 and r_2, respectively, as

$$FA_1 = (Q_1, \Sigma_1, \delta_1, q_1, f_1), \quad FA_2 = (Q_2, \Sigma_2, \delta_2, q_2, f_2).$$

Case 1: Construction of $L_1 + L_2$

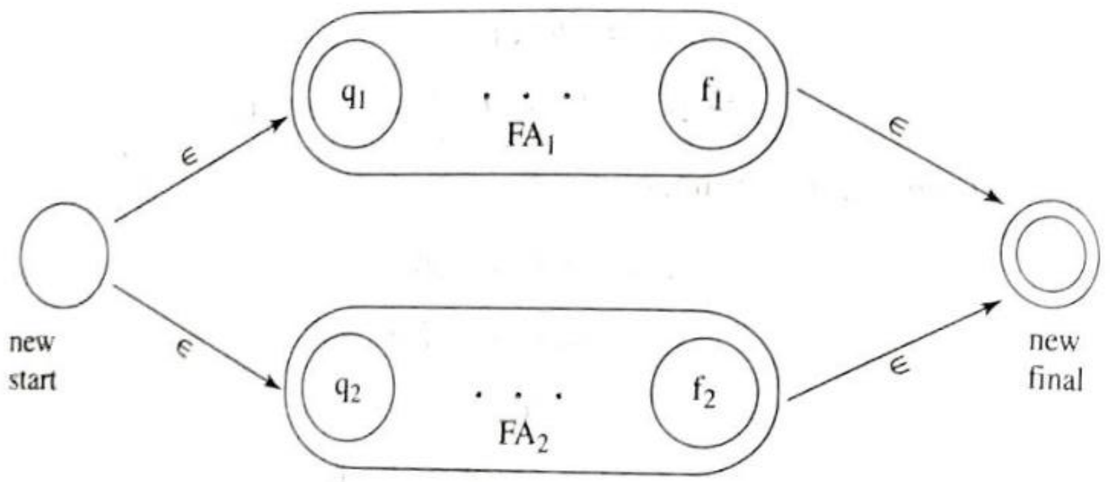


Figure 7.1. *Automaton for $L_1 + L_2$.*

Case 2: Construction of L_1L_2

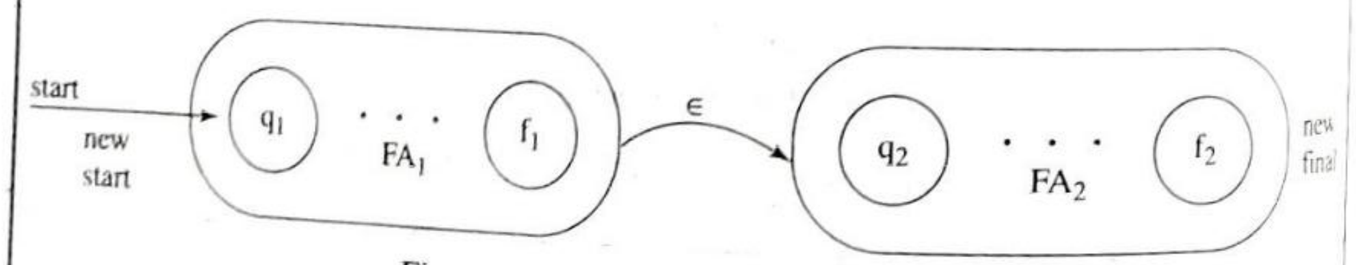


Figure 7.2. *Automaton for L_1L_2.*

Case 3: Construction of L_1^*

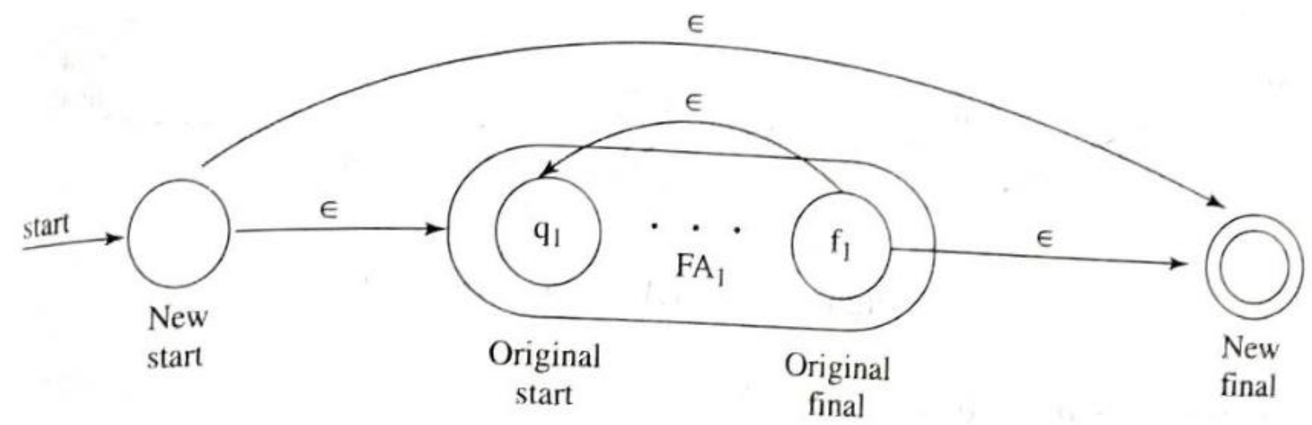


Figure 7.3. *Automaton for L_1^*.*

EXAMPLE 7.1.2: (method 2)

Let $\Sigma = \{a, b\}$, L_1 = all words ending with a.

L_2 = all words containing substring aa.

$r_1 = (a + b)^*a$, $r_2 = (a + b)^*aa(a + b)^*$.

Then,

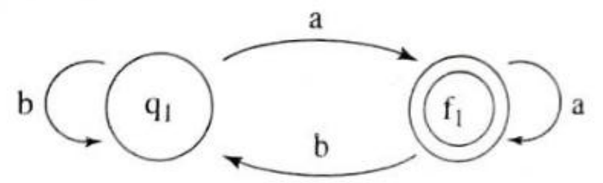


Figure 7.4. *FA_1 for r_1.*

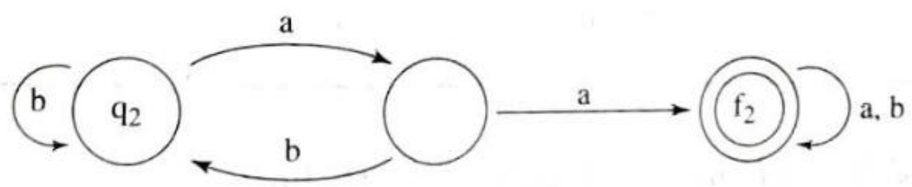


Figure 7.5. *FA_2 for r_2.*

(i) $r_1 + r_2 = (a + b)^*a + (a + b)^*aa(a + b)^*$.

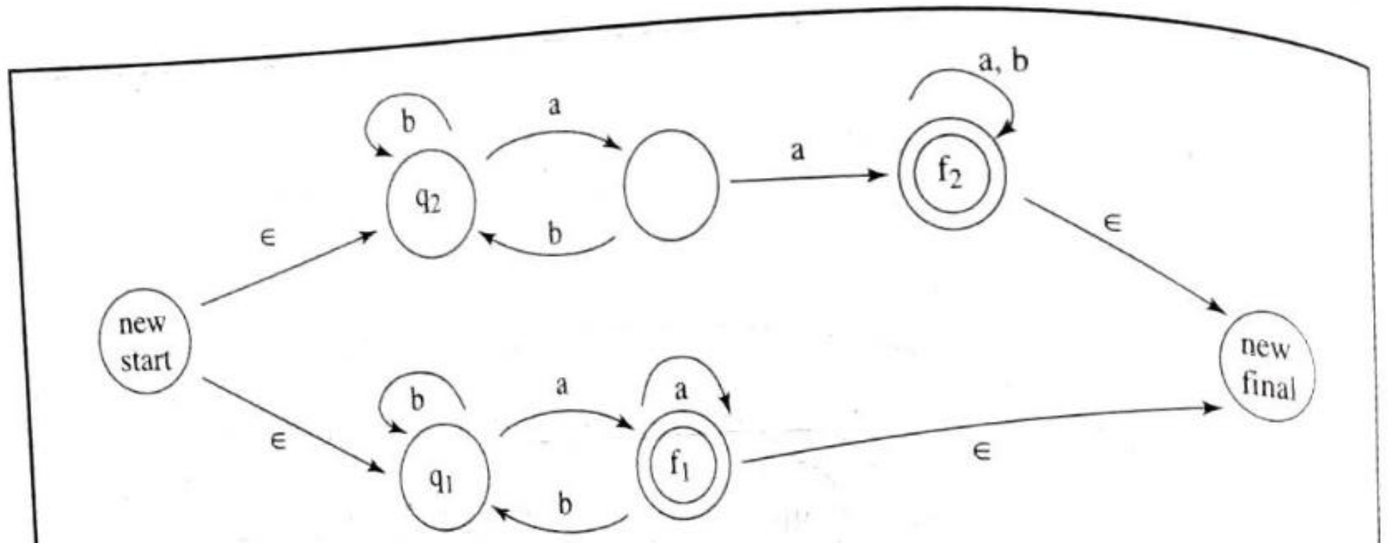


Figure 7.6. *FA for* $r_1 + r_2$.

(ii) $r_1 r_2 = (a+b)^* a \cdot (a+b)^* aa(a+b)^*$

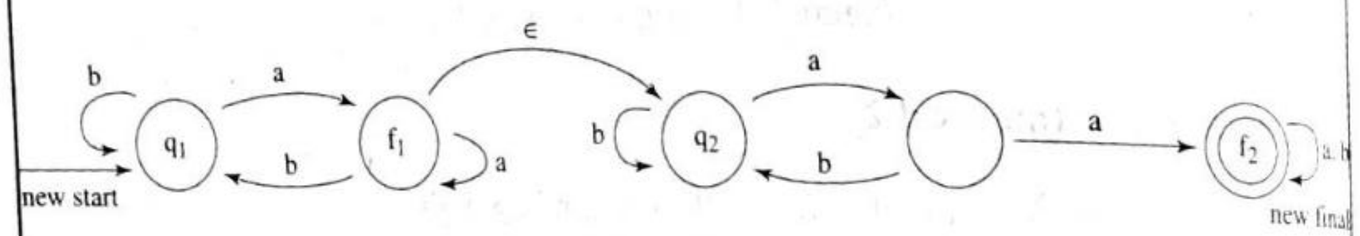


Figure 7.7. *FA for* $r_1 r_2$.

(iii) $r_2^* = ((a+b)^* aa(a+b)^*)^*$

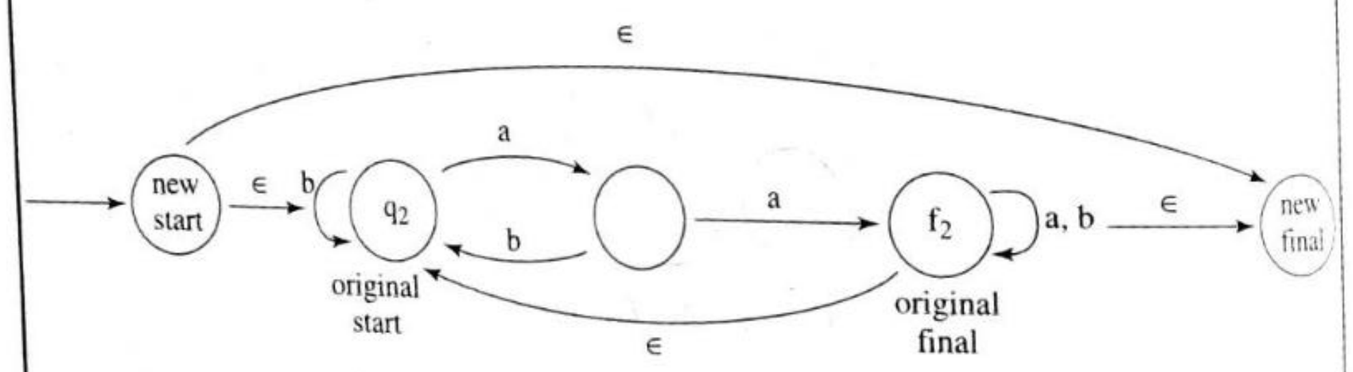


Figure 7.8. *FA for* r_2*.

7.1.2 *Complementation of a Regular Language*

As discussed earlier, complement of a language L is a language $\bar{L}$ such that,

$$\bar{L} = \Sigma^* - L.$$

For example, if $\Sigma = \{a, b\}, L = \{a, b, aa\}$ then $\bar{L}$ is as follows:

$$\bar{L} = \{\epsilon, a, b, aa, bb, aaa, bbb, ba, bab, abbab \ldots\} - \{a, b, aa\}.$$

$$\bar{L} = \{\epsilon, bb, aaa, bbb, ba, bab, abab, \ldots\}.$$

Theorem II

Regular languages are closed under complementation.

TPT: If L is a regular language and $L \sqsubseteq \Sigma^*$, then $\bar{L} = \Sigma^* - L$, is also regular.

Proof:

Suppose, L is a regular language over the alphabet Σ, then there is a DFA, $D = (Q, \Sigma, \delta, q_0, F)$, accepting $L = L(D)$.

To accept $\bar{L} = \Sigma^* - L$, complement the final states of D. For this, consider another DFA,

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

where, $F' = Q - F$, i.e., D and D' differ only in their final states.

Now, if 'x' is an input string/word, then $x \in L(D')$.

$$\Rightarrow \delta(q_0, x) \in F'$$
$$\Rightarrow \delta(q_0, x) \in Q - F$$
$$\Rightarrow \delta(q_0, x) \notin F$$
$$\Rightarrow x \notin L$$
$$\Rightarrow x \in \bar{L}$$
$$\Rightarrow x \in \Sigma^* - L.$$

Therefore, $L(D') = \overline{L} = \Sigma^* - L$.

Hence, $\overline{L}$ is a regular language since it is accepted by a DFA.

This can be explained as follows:

Consider a DFA D, accepting a language L which is regular. Now, create a DFA D', that accepts the language $\overline{L}$, as follows:

a. D' has same states and arcs as D.
b. Every final state of D becomes a non-final state in D'.
c. Every non-final state of D becomes a final state in D'.

$$\Rightarrow \text{ nonfinal states } \longleftrightarrow \text{ final states}$$

Thus, the resulting DFA, D' accepts $\overline{L}$, which is a regular language.

EXAMPLE 7.1.3: (Complementation)

If Σ = {a,b} and L = all words with length of atleast 2 and second letter as b and $\overline{L}$ = all words with length less than 2 or second letter as a, then the change of all non-final to final and all final to non-final states of DFA, D gives D', that accepts $\overline{L}$.

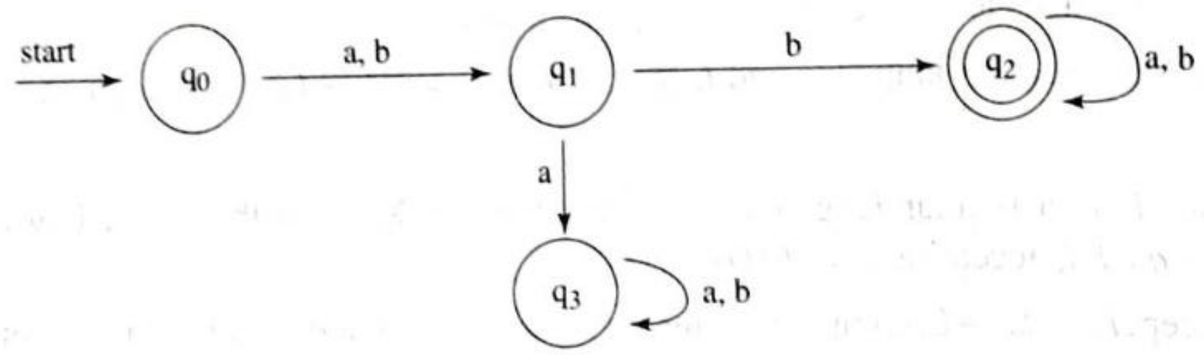


Figure 7.9. *DFA D*

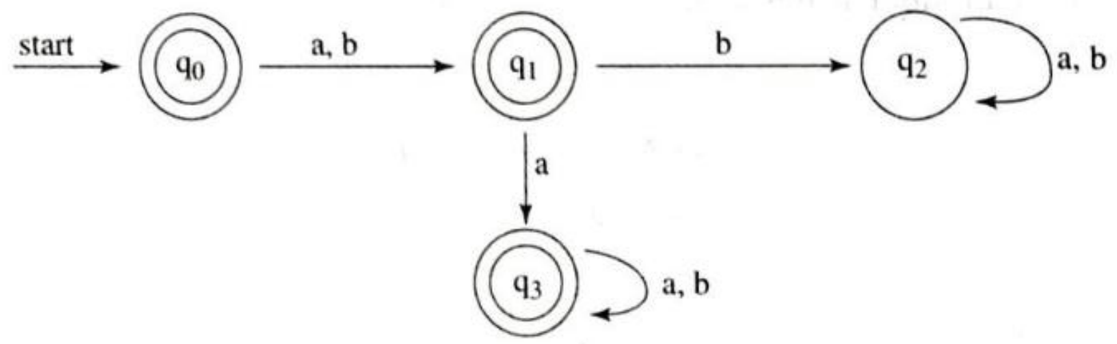


Figure 7.10. *DFA D'*

EXAMPLE 7.1.4: Give examples for closure properties of regular languages.
Let $L_1 = \{a^n b\}$, $L_2 = \{ba\}$. The transitions of L_1 and L_2 are shown in figure 7.11 and figure 7.12 respectively.

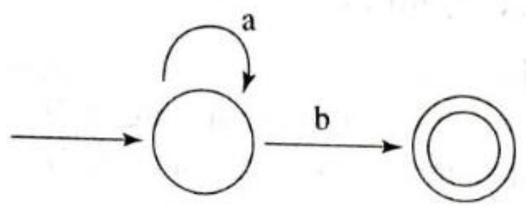


Figure 7.11. *M_1 for L_1 i.e. $L_1 = L_1(M_1)$*

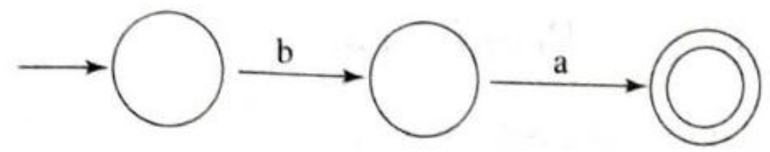


Figure 7.12. *M_2 for L_2 i.e. $L_2 = L_2(M_2)$*

- NFA for $L_1 \cup L_2$ is:

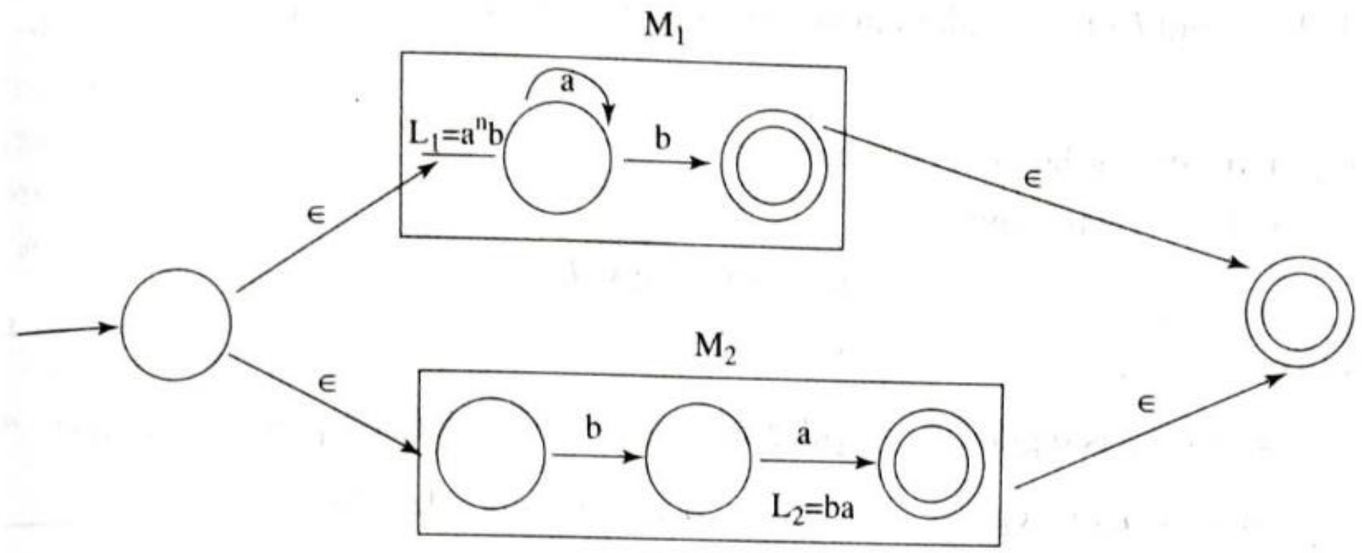


Figure 7.13. $L_1 \cup L_2 = \{a^n b \cup ba\}$

- NFA for $L_1 \cdot L_2$ is:

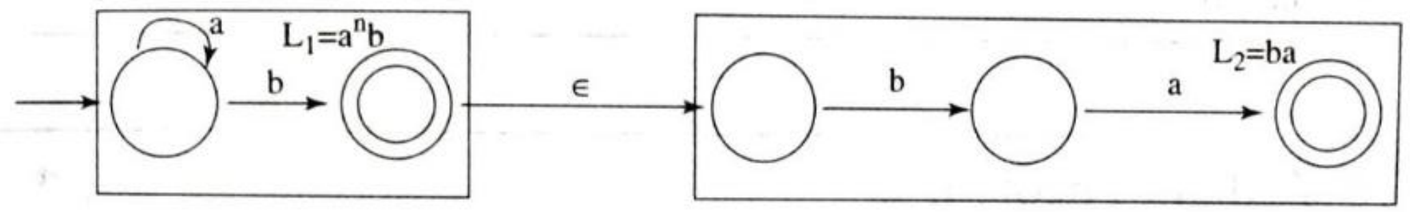


Figure 7.14. $L_1 \cdot L_2 = \{a^n \cdot b\} \cdot \{ba\} = \{a^n bba\}$.

- NFA for L_1^* is:

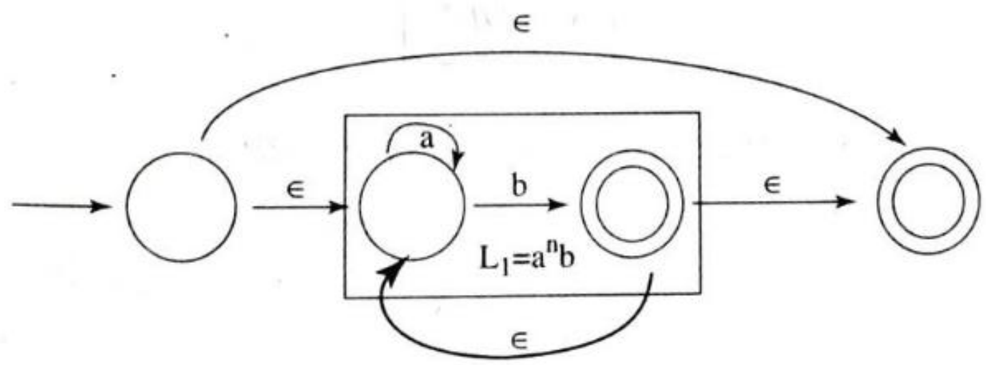


Figure 7.15. $L_1^* = \{a^n b\}^*$

7.1.3 *Intersection of Two Regular Languages*

Intersection of two regular languages L_1 and L_2 is defined as $L_1 \cap L_2$.

Theorem III

Regular Languages are closed under intersection.

T.P.T: If L_1 and L_2 are regular languages, then $L_1 \cap L_2$ is also regular.

Proof:

Given : Two regular languages L_1 and L_2.

Applying de Morgan's law,

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}.$$

Since L_1, L_2 is regular,

$\Rightarrow \overline{L}_1, \overline{L}_2$ is regular ($\because$ regular sets are closed under complementation)

$\Rightarrow \overline{L}_1 \cup \overline{L}_2$ is regular ($\because$ regular sets are closed under union)

$\Rightarrow \overline{\overline{L_1} \cup \overline{L_2}}$ is regular ($\because$ regular sets are closed under complementation)

$\Rightarrow L_1 \cap L_2$ is regular (by de Morgan's Law).

Hence, proved.

7.2 Arden's Theorem

Let P and Q be two regular expressions over Σ, and if P does not contain ϵ, then $R = Q + RP$ has a unique solution given by

$$\boxed{R = QP^*.}$$

If $Q = \epsilon$ then, $R = Q + RP$

$\Rightarrow R = \epsilon + RP$

$\Rightarrow R = \epsilon P^*$

$\Rightarrow R = P^*$.

This theorem is used to find a RE, which can be recognised by transitions.

7.3 Equivalence of Finite Automata and Regular Expressions

It is interesting to note that regular expressions and finite automata are equivalent in their descriptive power, although superficially they appear to be rather different. However, any regular expression can be converted into a finite automaton that recognises the language it describes and vice-versa. At this point, it can be recalled that a regular language is one that is recognised by some finite automaton.

The languages accepted by finite automata, are the languages denoted by regular expressions. In other words, we can say that regular expressions define the class of languages that are accepted by finite automata. This implies that:

a. every language accepted by a finite automaton can also be defined by a regular expression,
b. for every regular expression, there is an equivalent NFA with ϵ-transitions.

7.4 Cycle of Constructions

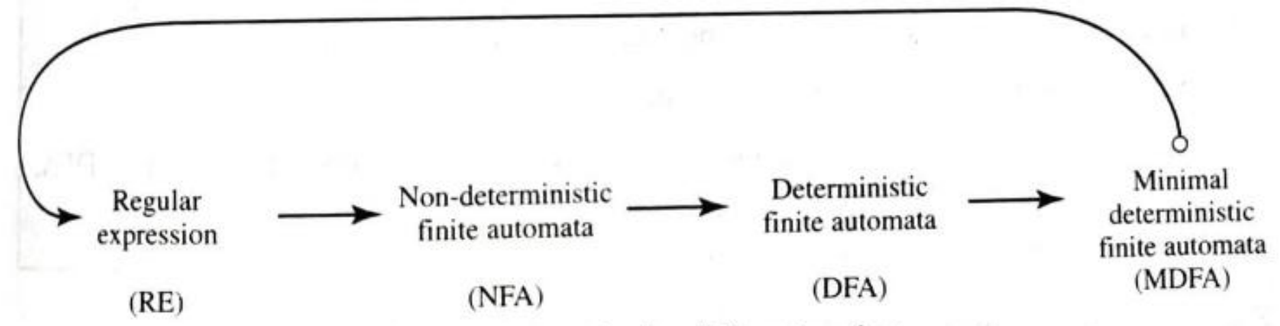


Figure 7.16. *Cycle of Construction.*

- NFA → DFA (subset construction)
- DFA → Minimal DFA
- RE → NFA (build an NFA for each term; combine them with ϵ-moves)
- DFA → RE

7.5 Equivalence of DFA and Regular Expressions

If L is accepted by a DFA, then L can be expressed by a regular language. In other words, for any given DFA, we can obtain a RE and vice-versa.

Following are the two methods of constructing RE from a given DFA.

a. By solving equations.
b. By using transition diagrams.

7.5.1 *Construction of RE from given DFA by Solving Equations*

The following are the assumptions made, regarding the transition diagrams:

a. the diagram must have ϵ-moves.
b. It has an initial state, say q_1.
c. It has vertices $q_1, q_2, \ldots q_n$.
d. q_i is the RE representing the set of strings, accepted by the system through q_i (the final state).

Following are the steps required to construct a RE:

Step-1: For each of the states $q_1, \ldots q_n$ in DFA, write down the equations by considering all edges, that enter into that state.

Step-2: For the initial state of DFA, the equation is added with ϵ.

Step-3: Compute the equation for each state.

Step-4: Substitute the results of each state equation into the final state equation of DFA, to get a RE for DFA.

EXAMPLE 7.5.1: Construct a RE corresponding to the DFA, represented by table 7.1. q_1 is both the initial and the final state. Transition table is given below:

Q \ Σ	Present inputs 0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_1	q_2

Table 7.1 *State Transition Table*

Solution:

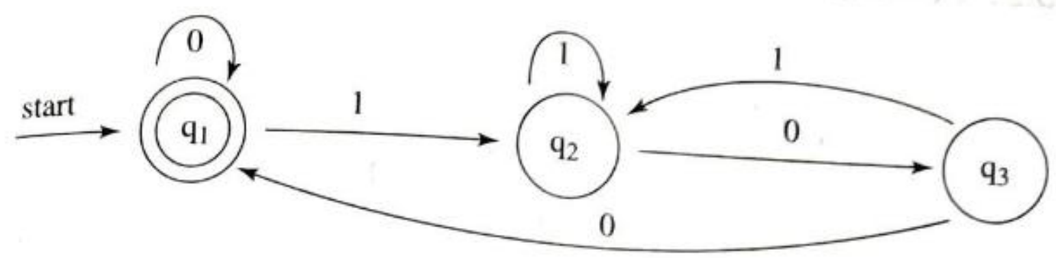


Figure 7.17. *Transition Diagram.*

Now, the equations for each state can be written, by considering all edges that enter into that state.

Thus,

$$q_1 = q_1 0 + q_3 0 + \epsilon \quad (\epsilon, \text{ only for initial state})$$
$$q_2 = q_1 1 + q_2 1 + q_3 1$$
$$q_3 = q_2 0.$$

Substituting q_3 in q_2

$$\Rightarrow q_2 = q_1 1 + q_2 1 + q_2 01$$
$$= q_1 1 + q_2(1 + 01)$$
$$= q_1 1(1 + 01)^* \quad (\because R = Q + RP \text{ is } R = QP^*)$$

Now,

$$q_1 = q_1 0 + q_3 0 + \epsilon$$
$$= q_1 0 + q_2 0 \cdot 0 + \epsilon$$
$$= q_1 0 + q_1 1(1 + 01)^* \cdot 00 + \epsilon$$
$$= q_1(0 + 1(1 + 01)^* \cdot 00) + \epsilon$$
$$= \epsilon\,(0 + 1(1 + 01)^* \cdot 00)^*$$
$$(\because R = Q + RP \Rightarrow R = QP^*)$$
$$q_1 = (0 + 1(1 + 01)^* \cdot 00)^*$$

Since, q_1 is the final state, RE for DFA will be

$$RE = (0 + 1(1 + 01)^*00).$$

EXAMPLE 7.5.2: Construct a RE for the DFA in figure 7.18.

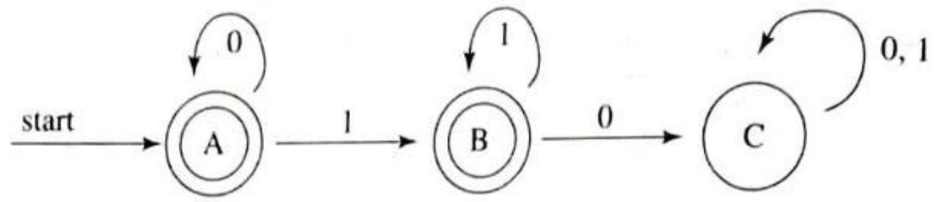


Figure 7.18. *Transition Diagram.*

Solution: Writing the equations for each state, by considering all edges that enter into that state:

$$A = A0 + \epsilon$$
$$B = A1 + B1$$
$$C = B0 + C(0+1)$$

Since,

$$R = Q + RP \Rightarrow R = QP^*.$$

Consider,

$$A = A0 + \epsilon, \quad \Rightarrow A = \epsilon 0^* \quad (\because R = A,\ Q = \epsilon,\ P = 0).$$
$$A = 0^*.$$

Now, substitute A in $B = A_1 + B_1$

$$\Rightarrow B = 0^*1 + B_1$$
$$B = 0^*11^* \qquad (\because R = Q + RP \Rightarrow R = QP^*).$$

Since, A and B are the final states in the above DFA, the addition of A and B gives,

$$\Rightarrow A + B = 0^* + 0^*11^*$$
$$= 0^*(\epsilon + 11^*)$$
$$= 0^*(\epsilon + 11^*)$$
$$= 0^*1^* \qquad (\because \epsilon + RR^* = R^*)$$
$$\Rightarrow RE = 0^*1^*.$$

EXAMPLE 7.5.3: Construct RE for the DFA in figure 7.19.

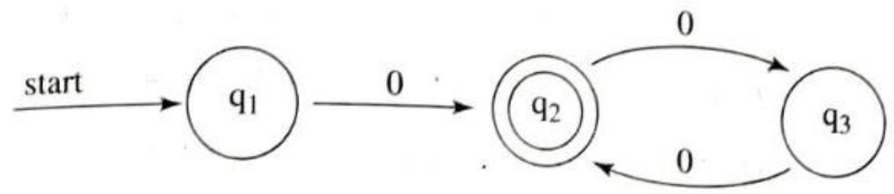


Figure 7.19. *Transition Diagram.*

Solution:

$$q_1 = \epsilon,\ q_2 = q_1 0 + q_3 0 \quad \text{and} \quad q_3 = q_2 0.$$

Computing q_2 :

$$q_2 = q_1 0 + q_3 0$$
$$q_2 = \epsilon\, 0 + q_2 0 \cdot 0$$
$$q_2 = \epsilon\, 0(00)^*$$
$$\Rightarrow RE = 0 \cdot (00)^*.$$

EXAMPLE 7.5.4: Construct RE for the DFA in figure 7.20.

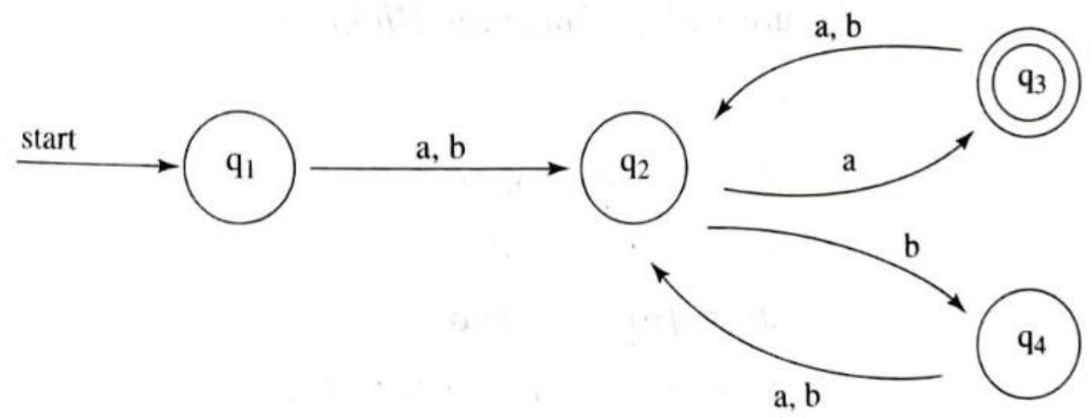


Figure 7.20. *Transition Diagram.*

Solution:

$$q_1 = \epsilon$$
$$q_2 = q_1(a+b) + q_3(a+b) + q_4(a+b)$$
$$q_3 = q_2 a$$
$$q_4 = q_2 b$$

Now, computing q_2: (because computing for q_3 leads to complications).

$$\Rightarrow q_2 = q_1(a+b) + q_3(a+b) + q_4(a+b)$$
$$q_2 = \in (a+b) + q_2a(a+b) + q_2b(a+b)$$
$$q_2 = \in (a+b) + q_2(a(a+b) + b(a+b))$$
$$q_2 = \in (a+b) \cdot (a(a+b) + b(a+b))^*$$
$$\Rightarrow RE = (a+b) \cdot (a(a+b))^* \cdot (b(a+b))^*.$$

EXAMPLE 7.5.5: Construct RE for the DFA given in figure 7.21.

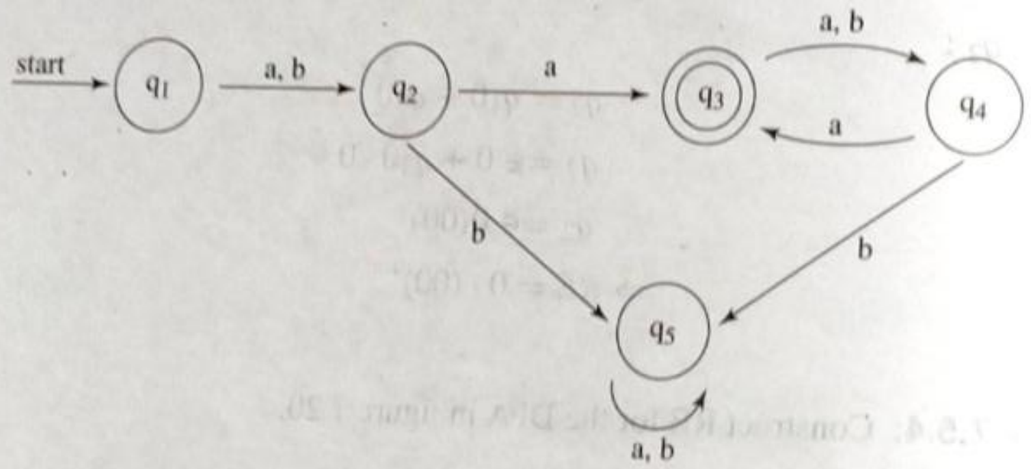


Figure 7.21. *Transition Diagram.*

Solution:

$$q_1 = \in, \quad q_2 = q_1(a+b).$$
$$q_3 = q_2a + q_4 \cdot a,$$
$$q_4 = q_3(a+b), \text{ and}$$
$$q_5 = q_2b + q_4b + q_5(a+b).$$

Since $q_1 = \in$, $\Rightarrow$ $q_2 = \in (a+b)$

Now, computing q_3:

$$q_3 = (a+b)a + q_4 \cdot a$$
$$q_3 = (a+b)a + q_3 \cdot (a+b)a$$
$$q_3 = (a+b)a \cdot (a(a+b))^*$$
$$\Rightarrow RE = (a+b) \cdot a(a(a+b))^*.$$

EXAMPLE 7.5.6: Construct a RE for the given transition diagram.

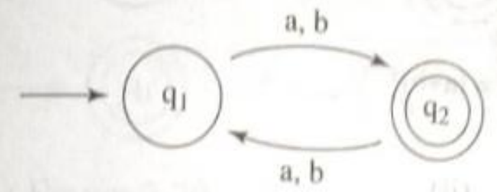


Figure 7.22. *Transition Diagram.*

Solution:

$$q_1 = q_2(a+b) + \in$$
$$q_2 = q_1(a+b)$$

Computing for q_2:

$$q_2 = q_1(a+b)$$
$$q_2 = q_2(a+b)(a+b) + \in$$
$$q_2 = ((a+b)(a+b))^* \in$$
$$\Rightarrow RE = ((a+b)(a+b))^*.$$

7.5.2 *Construction of RE from the given DFA by using the Transition Diagram*

The following are some transition diagrams and their equivalent REs.

a. If a transition diagram has no final state, then $RE = \phi$, i.e., FA does not accept any string.

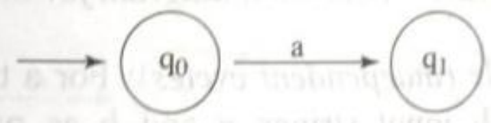


Figure 7.23. $RE = \phi$

b. If a transition diagram has the same start and final states, then $RE = \in$, i.e., FA accepts one of the strings as $\in$.

Figure 7.24. $RE = \in$

c. For a transition diagram, with transition to itself (say for the input string b or (a or b)), then $RE = b^*$ *or* $RE = (a+b)^*$.

Now, computing q_2: (because computing for q_3 leads to complications).

$$\Rightarrow q_2 = q_1(a+b) + q_3(a+b) + q_4(a+b)$$
$$q_2 = \epsilon\,(a+b) + q_2a(a+b) + q_2b(a+b)$$
$$q_2 = \epsilon\,(a+b) + q_2(a(a+b) + b(a+b))$$
$$q_2 = \epsilon\,(a+b)\cdot(a(a+b) + b(a+b))^*$$
$$\Rightarrow RE = (a+b)\cdot(a(a+b))^*\cdot(b(a+b))^*.$$

EXAMPLE 7.5.5: Construct RE for the DFA given in figure 7.21.

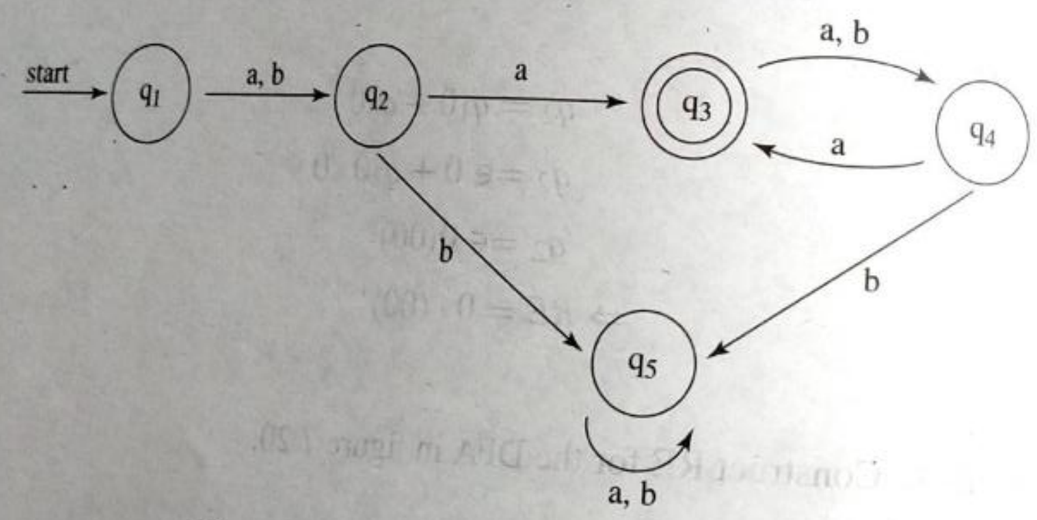


Figure 7.21. *Transition Diagram.*

Solution:

$$q_1 = \epsilon, \quad q_2 = q_1(a+b).$$
$$q_3 = q_2a + q_4\cdot a,$$
$$q_4 = q_3(a+b), \text{ and}$$
$$q_5 = q_2b + q_4b + q_5(a+b).$$

Since $q_1 = \epsilon$, $\Rightarrow$ $q_2 = \epsilon\,(a+b)$

Now, computing q_3:

$$q_3 = (a+b)a + q_4\cdot a$$
$$q_3 = (a+b)a + q_3\cdot(a+b)a$$
$$q_3 = (a+b)a\cdot(a(a+b))^*$$
$$\Rightarrow RE = (a+b)\cdot a(a(a+b))^*.$$

EXAMPLE 7.5.6: Construct a RE for the given transition diagram.

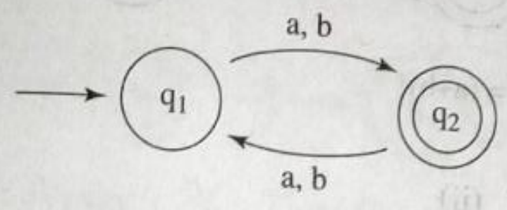


Figure 7.22. *Transition Diagram.*

Solution:

$$q_1 = q_2(a+b) + \epsilon$$
$$q_2 = q_1(a+b)$$

Computing for q_2:

$$q_2 = q_1(a+b)$$
$$q_2 = q_2(a+b)(a+b) + \epsilon$$
$$q_2 = ((a+b)(a+b))^*\,\epsilon$$
$$\Rightarrow RE = ((a+b)(a+b))^*.$$

7.5.2 *Construction of RE from the given DFA by using the Transition Diagram*

The following are some transition diagrams and their equivalent REs.

a. If a transition diagram has no final state, then $RE = \phi$, i.e., FA does not accept any string.

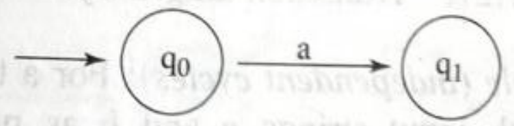


Figure 7.23. $RE = \phi$

b. If a transition diagram has the same start and final states, then $RE = \epsilon$, i.e., FA accepts one of the strings as ϵ.

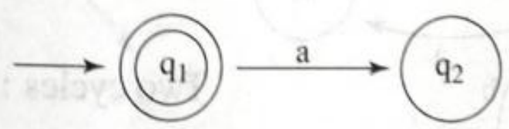


Figure 7.24. $RE = \epsilon$

c. For a transition diagram, with transition to itself (say for the input string b or (a or b)), then $RE = b^*$ *or* $RE = (a+b)^*$.

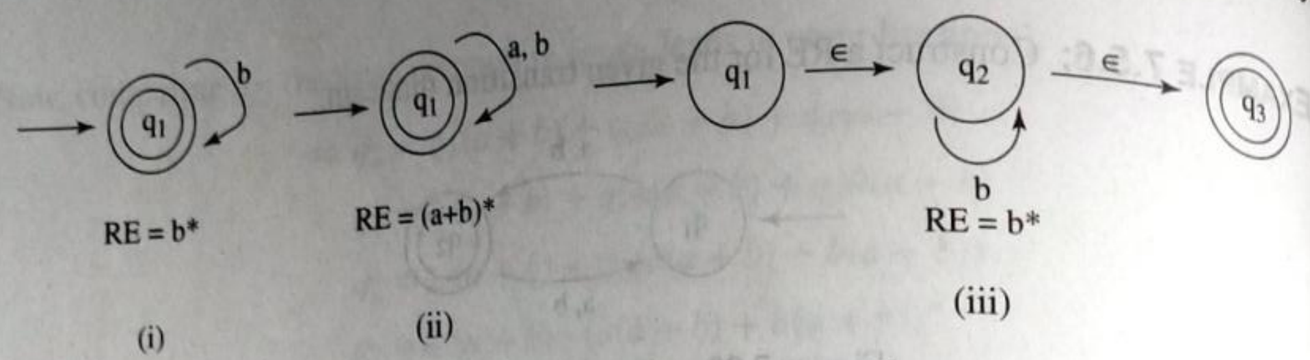


Figure 7.25. *Transition Diagram with Transition to Itself.*

d. A transition diagram, which goes to the final state with input strings a and b, the $RE = ab$.

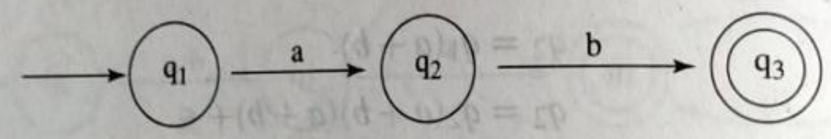
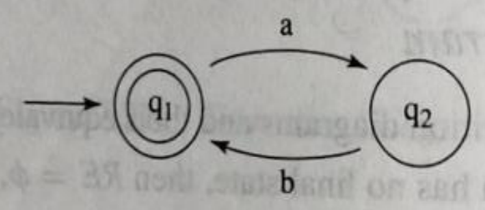


Figure 7.26. $RE = ab$.

e. Transition diagram with cycle.

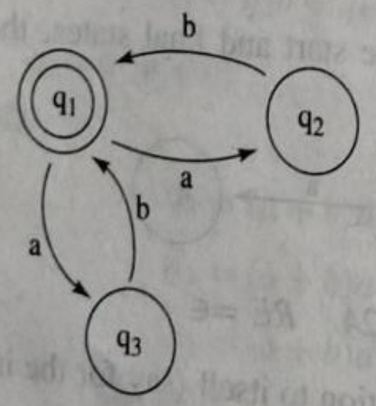
- *One-cycle:* For a transition diagram with one cycle, for the input (say a or b), the $RE = (ab)^*$.



One cycle: $q_1 - q_2 - q_1$
$RE = (ab)^*$

Figure 7.27. *Transition diagram for one cycle*

- *More than One cycle (independent cycles):* For a transition diagram with more than one cyle, with input strings a and b as part of the cycle, the RE is $RE = (ab)^* + (ab)^* + \ldots$ or $RE = (ab + ab + \ldots)^*$.



Two cycles : (i) $q_1 - q_2 - q_1$
(ii) $q_1 - q_3 - q_1$
$RE = (ab)^* + (ab)^*$ or $RE = (ab + ab)^*$

Figure 7.28. *Transition Diagram for more than One Cycle*

EXAMPLE 7.5.7: Construct the regular expression, from the DFA's, given below:

i.

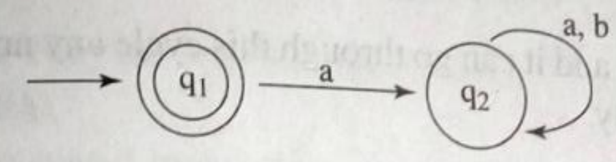


Figure 7.29. *State Diagram.*

Solution:

$$L(DFA) = \epsilon \text{ or } RE = \epsilon .$$

This DFA accepts ϵ because it goes from the start state to the final state, without reading any symbol or alphabet (i.e. by reading empty string ϵ). It accepts nothing else because any non-empty symbol would take it to state 2, which is not a final state and it stays there.

ii.

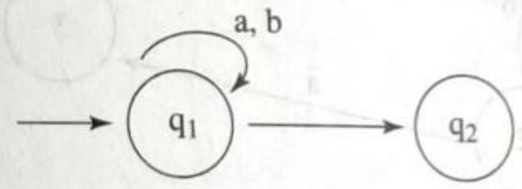


Figure 7.30. *State Diagram.*

Solution:

$$RE = \phi$$

This DFA does not accept any string because it has no accepting state. Thus, the RE is ϕ.

iii.

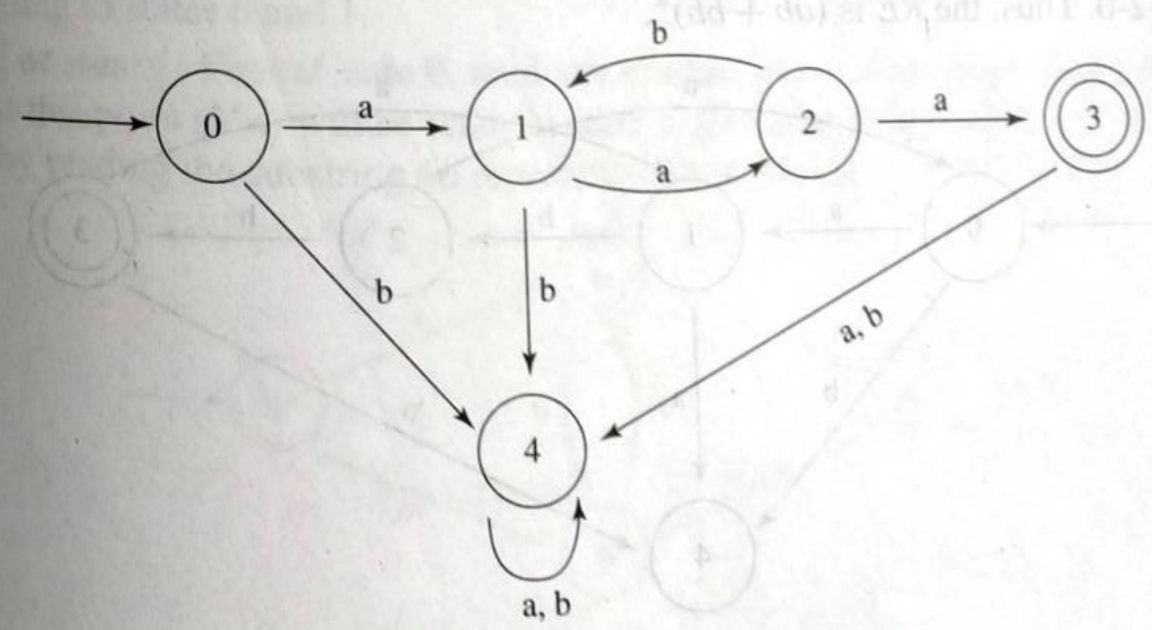


Figure 7.31. *State Diagram.*

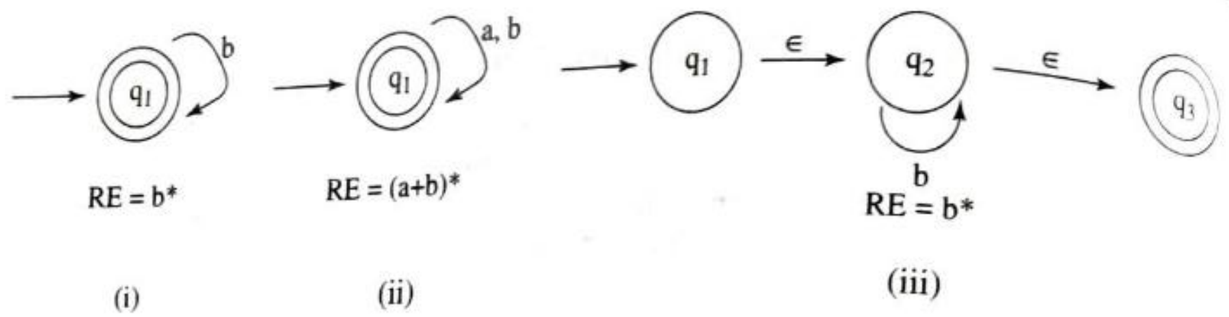


Figure 7.25. *Transition Diagram with Transition to Itself.*

d. A transition diagram, which goes to the final state with input strings a and b, the $RE = ab$.

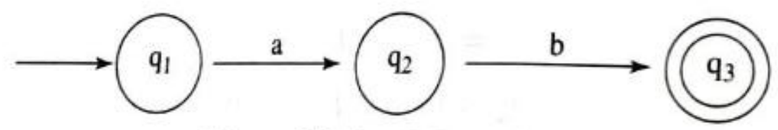
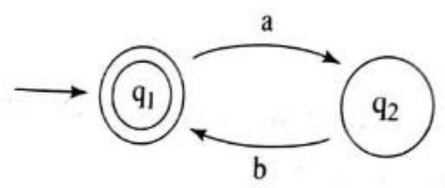


Figure 7.26. $RE = ab$.

e. Transition diagram with cycle.

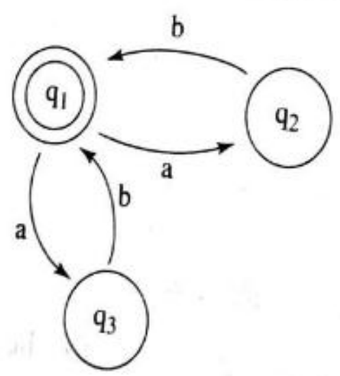
- *One-cycle:* For a transition diagram with one cycle, for the input (say a or b), the $RE = (ab)^*$.



One cycle: $q_1 - q_2 - q_1$
$RE = (ab)^*$

Figure 7.27. *Transition diagram for one cycle*

- *More than One cycle (independent cycles):* For a transition diagram with more than one cyle, with input strings a and b as part of the cycle, the RE is $RE = (ab)^* + (ab)^* + \ldots$ or $RE = (ab + ab + \ldots)^*$.



Two cycles : (i) $q_1 - q_2 - q_1$
(ii) $q_1 - q_3 - q_1$
$RE = (ab)^* + (ab)^*$ or $RE = (ab + ab)^*$

Figure 7.28. *Transition Diagram for more than One Cycle*

EXAMPLE 7.5.7: Construct the regular expression, from the DFA's, given below:

i.

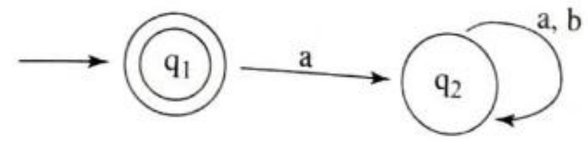


Figure 7.29. *State Diagram.*

Solution:

$$L(DFA) = \epsilon \text{ or } RE = \epsilon .$$

This DFA accepts ϵ because it goes from the start state to the final state, without reading any symbol or alphabet (i.e. by reading empty string ϵ). It accepts nothing else because any non-empty symbol would take it to state 2, which is not a final state and it stays there.

ii.

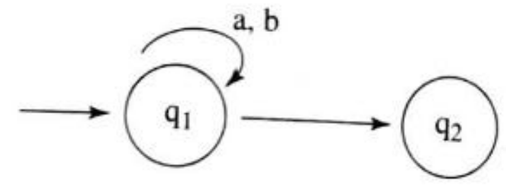


Figure 7.30. *State Diagram.*

Solution:

$$RE = \phi$$

This DFA does not accept any string because it has no accepting state. Thus, the RE is ϕ.

iii.

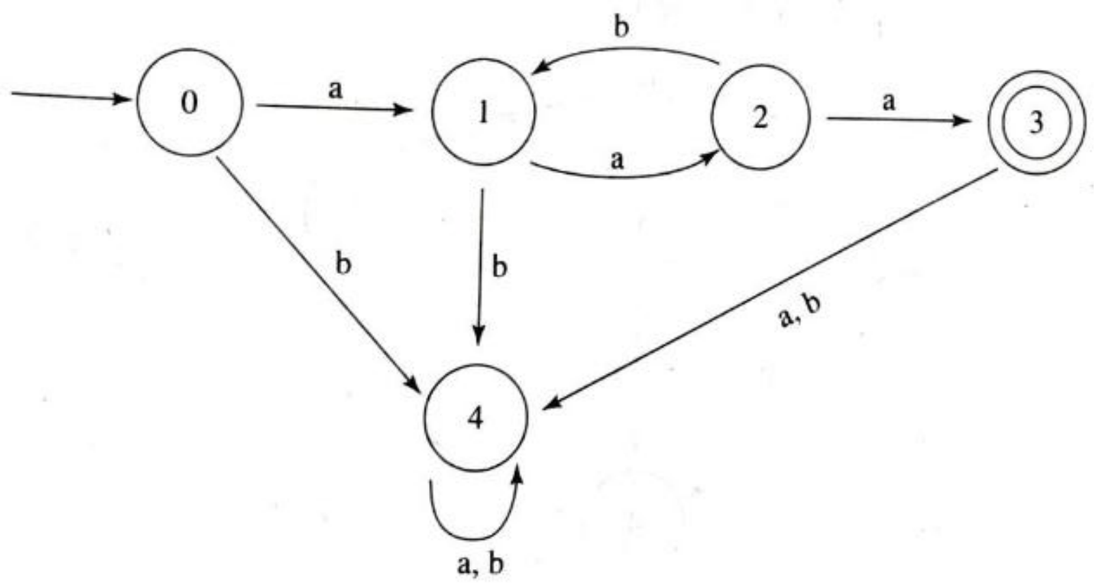


Figure 7.31. *State Diagram.*

Solution:

$$RE = a(ab)^*aa.$$

This DFA has a cycle 1-2-1 and it can go through this cycle any number of times, by reading the substring '*ab*' repeatedly.

To find the *RE*:

First from the start state it goes to state 1 by reading one '*a*'. Then, from state 1, it goes through the cycle 1-2-1, any number of times, by reading the substring *ab* any number of times and comes back to state 1. This is represented by $(ab)^*$. Then from state 1, it goes to state 2 and then to state 3, by reading *aa*. Thus, the *RE* is $a(ab)^*aa$.

iv.

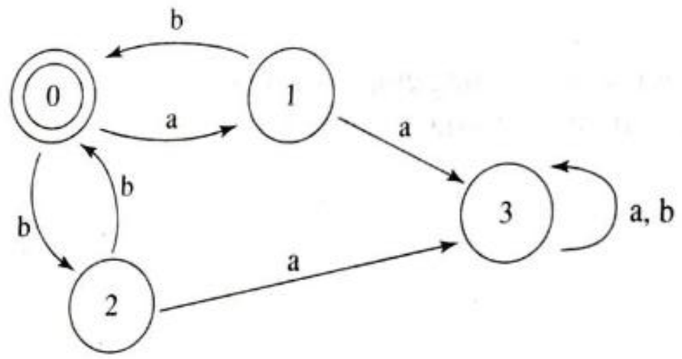


Figure 7.32. *State Diagram.*

Solution:

$$RE = (ab + bb)^*$$

This DFA has 2 independent cycles 0-1-0 and 0-2-0. It can move through these cycles any number of times, in any order, to reach the accepting state from the initial state, such as 0-1-0-2-0-2-0. Thus, the *RE* is $(ab + bb)^*$.

v.

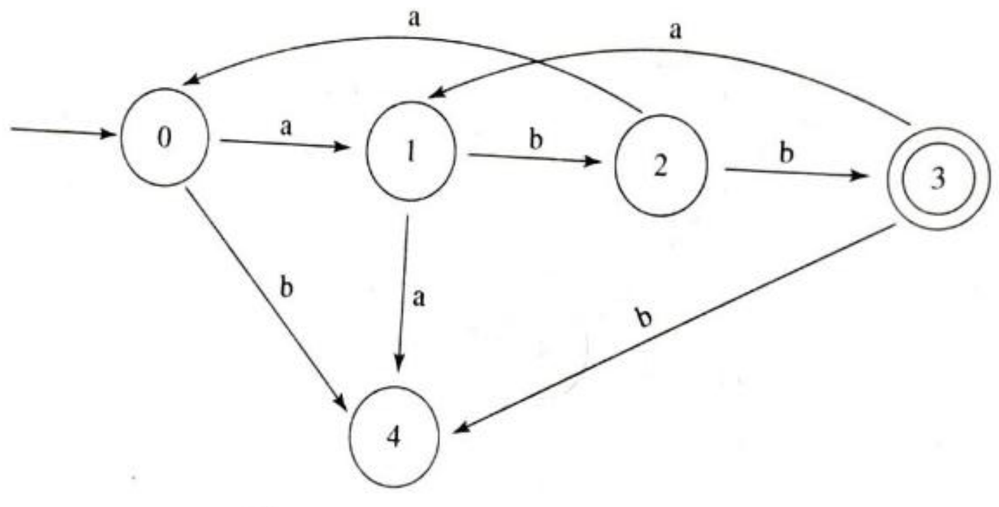


Figure 7.33. *State Diagram.*

Solution:

$$RE = a(bba + baa)^*bb.$$

This DFA has two cycles. They are 1-2-0-1 and 1-2-3-1.

To find the *RE* of this DFA:

First from state 0, go to state 1 by reading a (any other state, which is common to these cycles, such as state 2, can also be used instead of state 1). Then, from state 1 go through the 2 cycles 1-2-0-1 and 1-2-3-1, any number of times (in any order), by reading the substring *bba* and *baa* respectively. At this point, a substring $a(baa + bba)^*$ would have been read. Then, go from state 1 to state 2 and then to state 3 by reading *bb*. Thus, altogether $a(baa+bba)^*bb$ would have been read, when state 3 is reached from state 0.

vi.

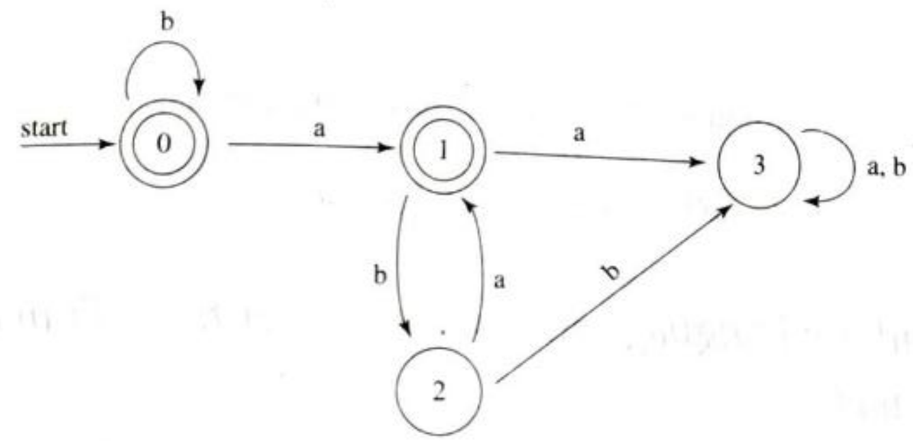


Figure 7.34. *State Diagram.*

Solution:

$$RE = b^* + b^*a(ba)^*$$

This DFA has 2 accepting states 0 and 1. Thus, the *RE* of this DFA is the union of the *RE* corresponding to states 0 and 1.

The RE at state 1: First at state 0, read any number of *b*'s, then go to state 1 by reading one '*a*'. At this point (b^*a) will be read. At state 1, go through the cycle 1-2-1, any number of times, by reading the substring *ba* repeatedly. Thus, the *RE* is $b^* + b^*a(ba)^*$.

vii.

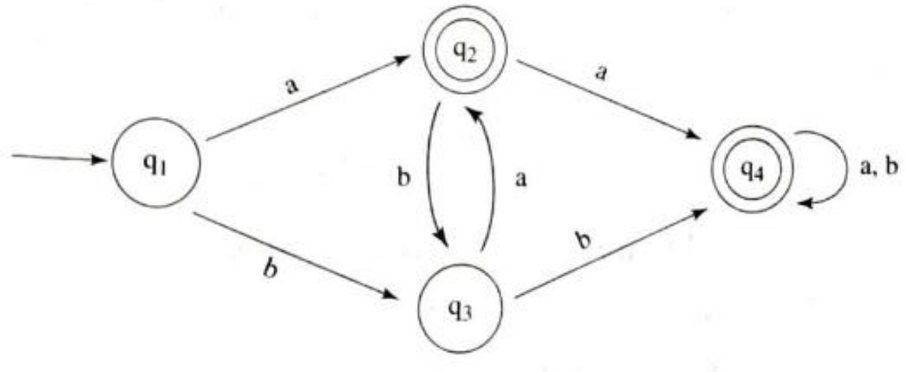


Figure 7.35. *State Transition Diagram.*

Solution:

$$RE = (aa + bb)(ba)^*(a + b)^*.$$

viii.

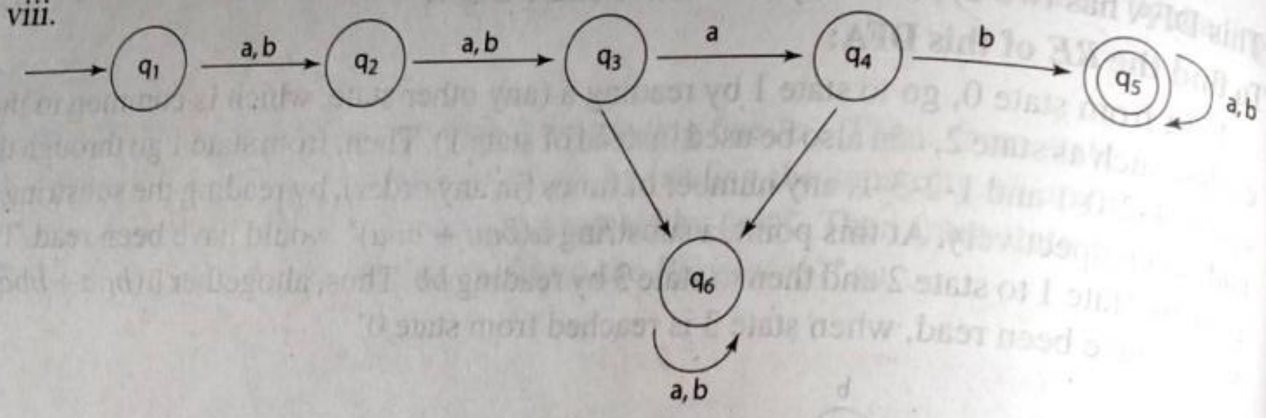


Figure 7.36. *State Transition Diagram.*

$$RE = (a + b)(a + b)ab(a + b)^*$$

7.5.3 To Find the Language Acceptence of FA from Transition Diagrams

As discussed in chapter 1, a FA is said to have accepted a string 's' where $s = w_1, w_2, \ldots w_n$, if there is a path in the transition diagram, such that, it

a. begins at a start state,
b. ends at an accepting state and
c. has a sequence of labels $w_1, w_2, \ldots, w_n$.

In general, to ascertain the language accepted by FA, first find the RE from the given FA, say 'M', then $L(M) = RE$.

EXAMPLE 7.5.8: What is the language accepted by the following DFA's?

i.

Figure 7.37. *State Diagram.*

Solution:

$$RE = 0.(00)^* \Rightarrow L(M) = 0(00)^*$$

i.e. a machine which accepts all zeroes.

ii.

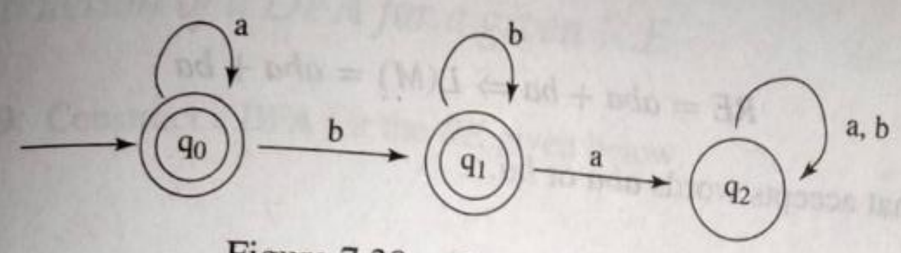


Figure 7.38. *State Diagram.*

Solution:

$$RE = a^*b^* \Rightarrow L(M) = a^*b^*$$

i.e. a machine that accepts any number of a's or any number of b's.

iii.

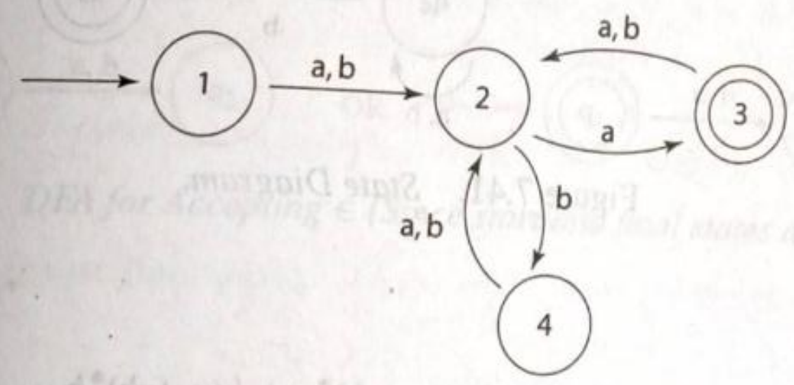


Figure 7.39. *State Diagram.*

Solution:

$$RE = (a + b) \cdot (a(a + b))^* \cdot (b(a + b))^*$$

$$\Rightarrow L(M) = (a + b)(a(a + b))^*(b(a + b))^*$$

iv.

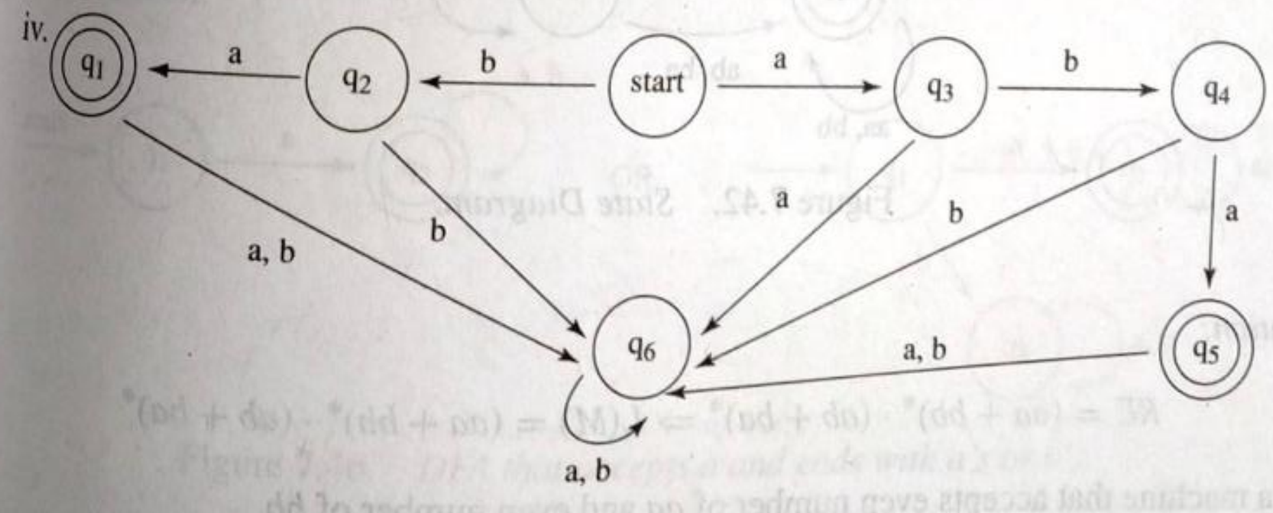


Figure 7.40. *State Diagram*

Solution:

$$RE = aba + ba \Rightarrow L(M) = aba + ba$$

i.e. a machine that accepts words *aba* or *ba*.

v.

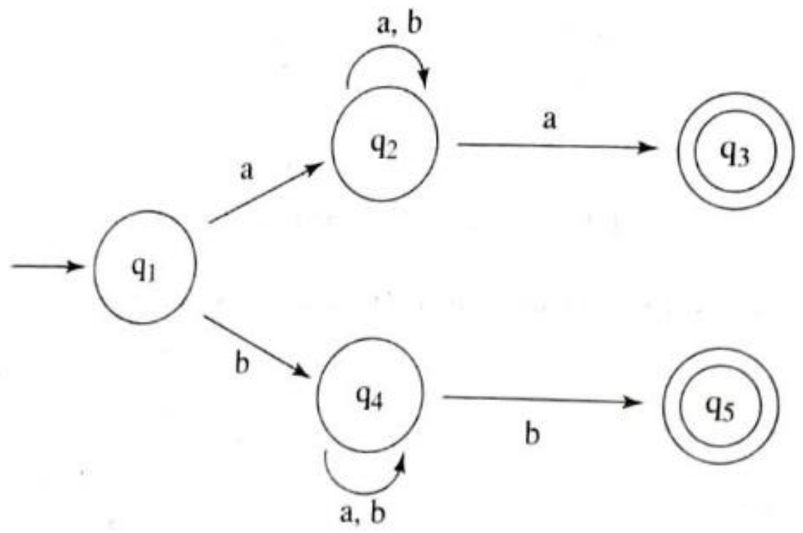


Figure 7.41. *State Diagram.*

Solution:

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

Thus, $L(M) = a(a+b)^*a + b(a+b)^*b$

i.e. a machine that accepts words that begin and end with '*a*' or begin and end with *b*.

vi.

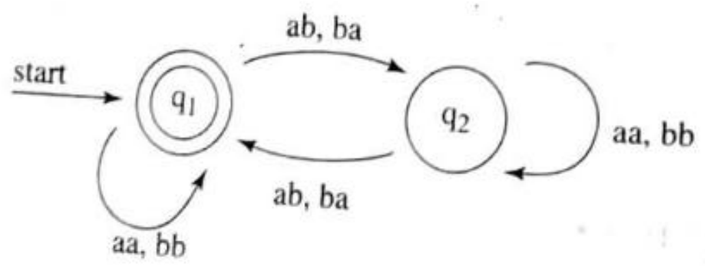


Figure 7.42. *State Diagram.*

Solution:

$$RE = (aa + bb)^* \cdot (ab + ba)^* \Rightarrow L(M) = (aa + bb)^* \cdot (ab + ba)^*$$

i.e. a machine that accepts even number of *aa* and even number of *bb*.

7.5.4 Construction of a DFA for a given R.E

EXAMPLE 7.5.9: Construct a DFA for the *RE* given below:

a. $RE = \phi$.
Solution:

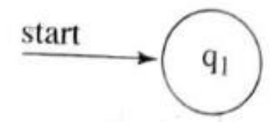


Figure 7.43. *DFA that Accepts Nothing; since it has no Final State.*

b. $RE = \in$ over $\Sigma = \{a, b\}$.
Solution:

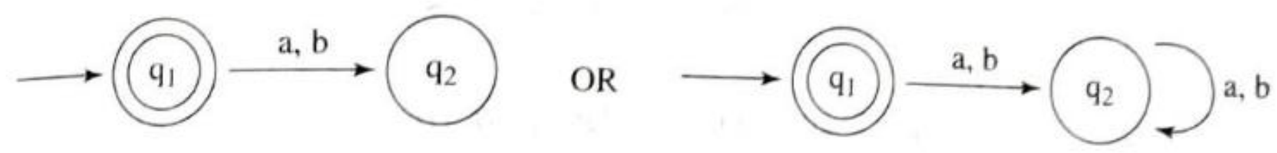


Figure 7.44. *DFA for Accepting ∈ (Since start and final states are the same).*

c. $RE = (a + b)^*$
Solution:

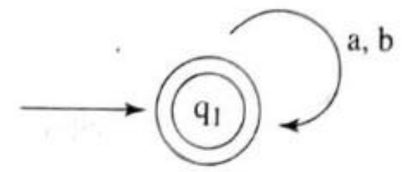


Figure 7.45. *DFA that accepts ∈ or a's or b's.*

d. $RE = a(a + b)^*$
Solution:

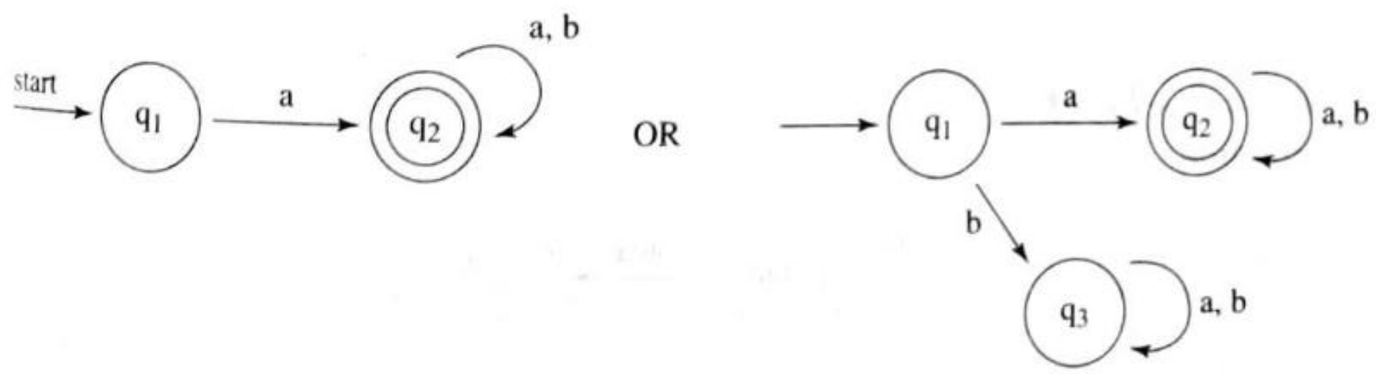


Figure 7.46. *DFA that accepts a and ends with a's or b's.*

e. ***RE = (a + b)(a + b)**** **over** $\Sigma = \{a, b\}$ **except** $\in$.
Solution:

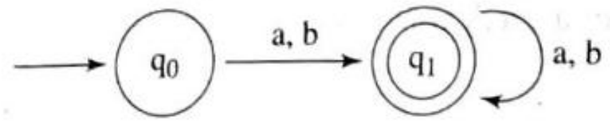


Figure 7.47. *State Diagram for DFA.*

f. ***RE = (a)*(ba)*b**** **over** $\Sigma = \{a, b\}$ **except** $\in$.
Solution:

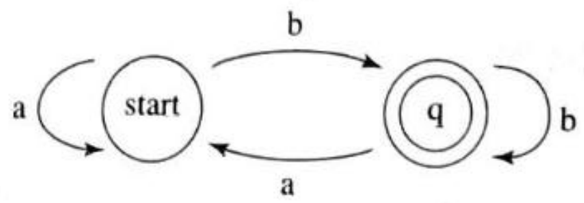


Figure 7.48. *DFA to Accept $a^*(ba)^*b^*$.*

g. ***RE =∈ + aaa + bbbb*** **over** $\Sigma = \{a, b\}$; **accept only words** $\in$, ***aaa, bbbb***
Solution:

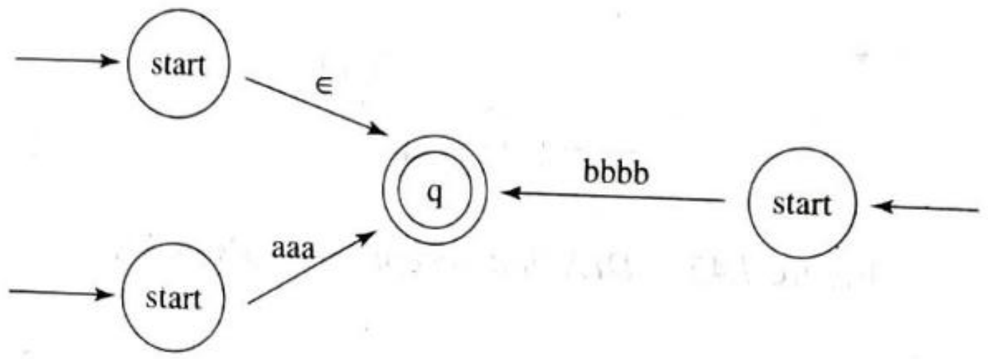


Figure 7.49. *State Diagram for DFA.*

h. ***RE = (a + b)* aba***
Solution:

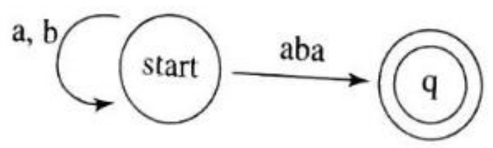


Figure 7.50. *DFA that Accepts Only those Words that end in aba.*

i. ***RE = (a + b)a[(a + b)a]****
Solution:

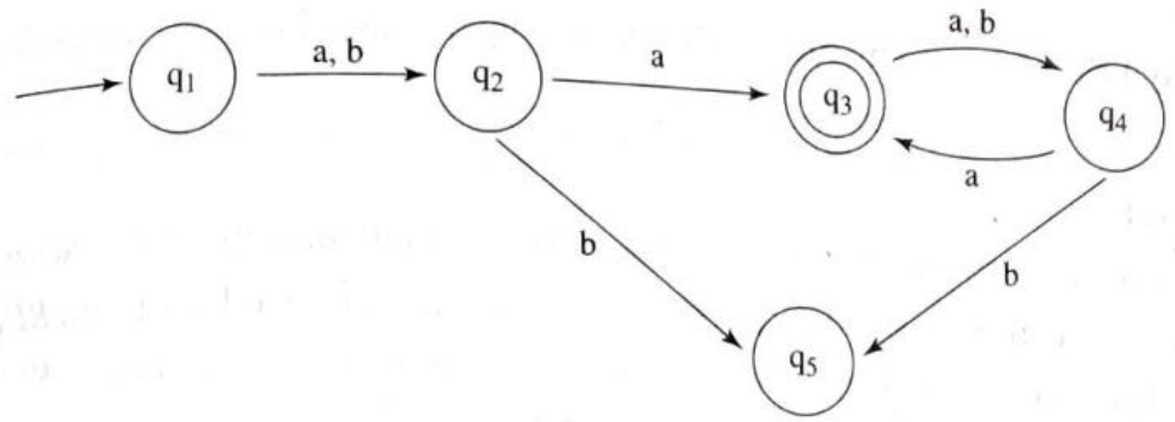


Figure 7.51. *DFA that accepts all words whose second letter is a, have an even number of letters in total and a in every even position.*

j. ***RE = baa + ab + abb.***
Solution:

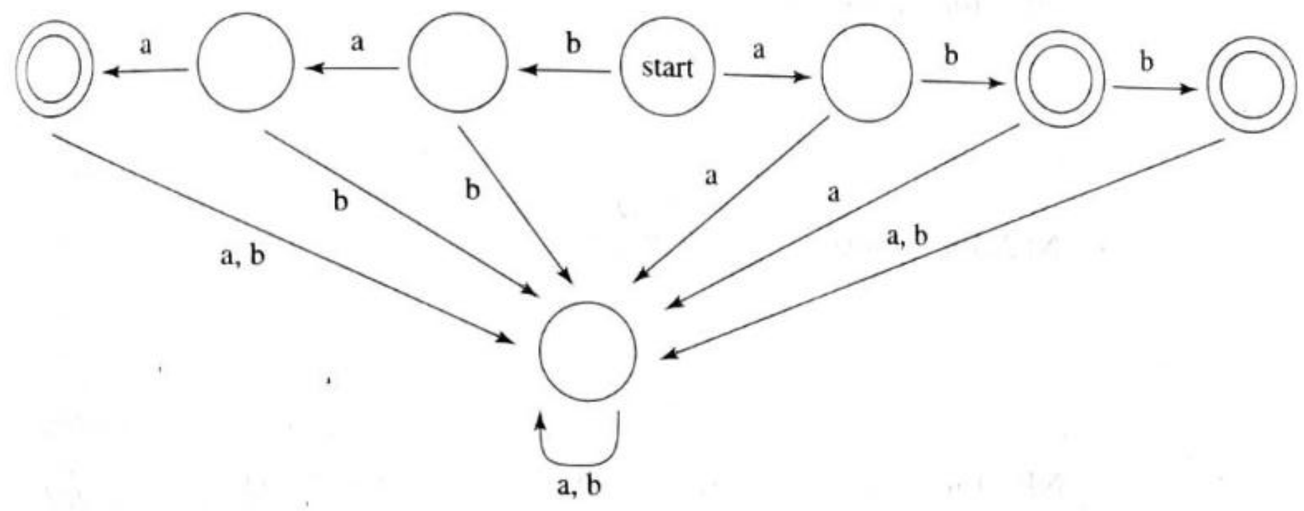


Figure 7.52. *DFA for RE = baa + ab + abb.*

k. ***RE = (a+b)* aa (a+b)****
Solution:

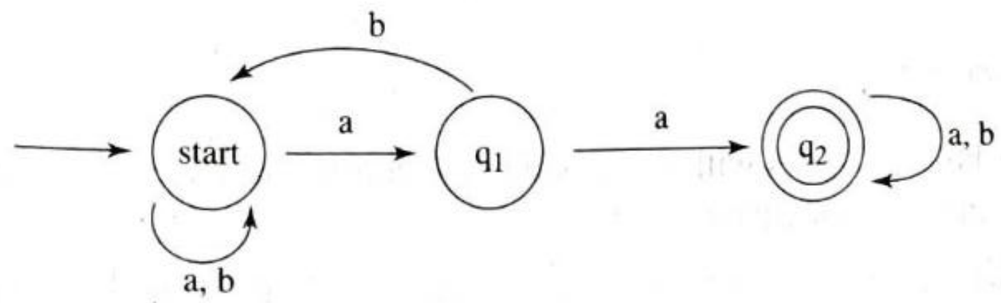


Figure 7.53. *DFA that accepts all words containing substring aa.*

7.6 Equivalence of NFA and Regular Expressions

From a given *RE*, we can construct an NFA accepting the same language, defined by a given *RE*.

Theorem IV

For every regular expression r, there exists an NFA with ϵ-transitions that accepts $L(r)$.

T.P.T: We can construct an NFA accepting the language, defined by a given *RE*.

Proof: This theorem can be proved by induction, on number of operators in *RE* 'r', i.e., there is an NFA N with ϵ- transition having one final state and no transition from that state. Hence, $L(N) = L(r)$.

Basis: (zero operators) – There are three possible *RE*s, having no operators, as shown in the figure 7.54:

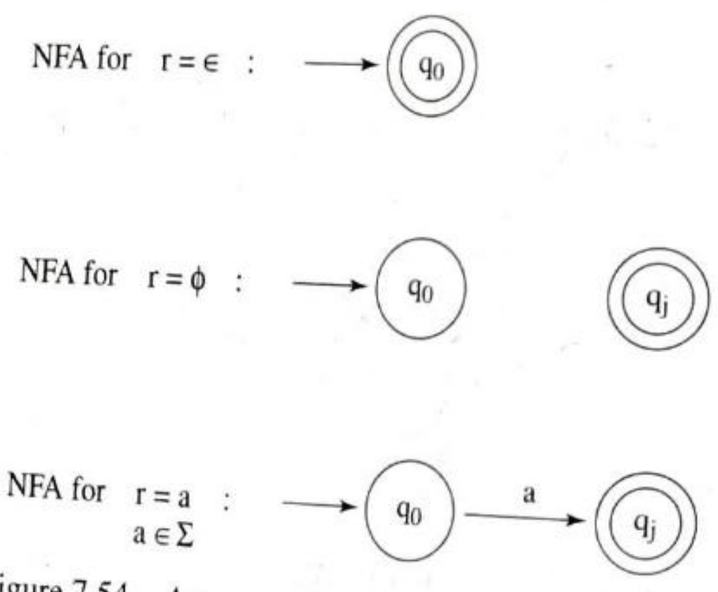


Figure 7.54. *Automata for* $r = \epsilon$ *or* $r = \phi$ *or* $r = a$.

There can be three different cases, depending on the form of *RE*. Assume, that the theorem is true for *RE* with some operators $i \geq 1$:

Case 1: $r = r_1 + r_2$.

Let N_1 and N_2 be two NFA's with ϵ-transitions accepting languages L_1 and L_2, defined by the REs r_1 and r_2 respectively.

$N_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\})$ and $N_2 = (Q_2, \Sigma_2, \delta_2, q_2, \{f_2\})$ with $L(N_1) = L(r_1)$. $L(N_2) = L(r_2)$.

Assume, that $Q_1 = Q_2 = \phi$, q_0 be a new initial state and f_0 be a new final state. Then, construct

$$N = (Q_1 \cup Q_2 \cup \{q_0, f_0\},\ \Sigma_1 \cup \Sigma_2,\ \delta, q_0, \{f_0\})$$

whose δ is defined by

a. $\delta(q_0, \epsilon) = \{q_1, q_2\}$.
b. $\delta(q, a) = \delta_1(q, a),\ q \in Q_1 - \{f_1\},\ a \in \Sigma_1 \cup \{\epsilon\}$.
c. $\delta(q, a) = \delta_2(q, a),\ q \in Q_2 - \{f_2\},\ a \in \Sigma_2 \cup \{\epsilon\}$.
d. $\delta(f_1, \epsilon) = \delta_1(f_2, \epsilon) = \{f_0\}$.

The NFA for the above transition is given in figure 7.55.

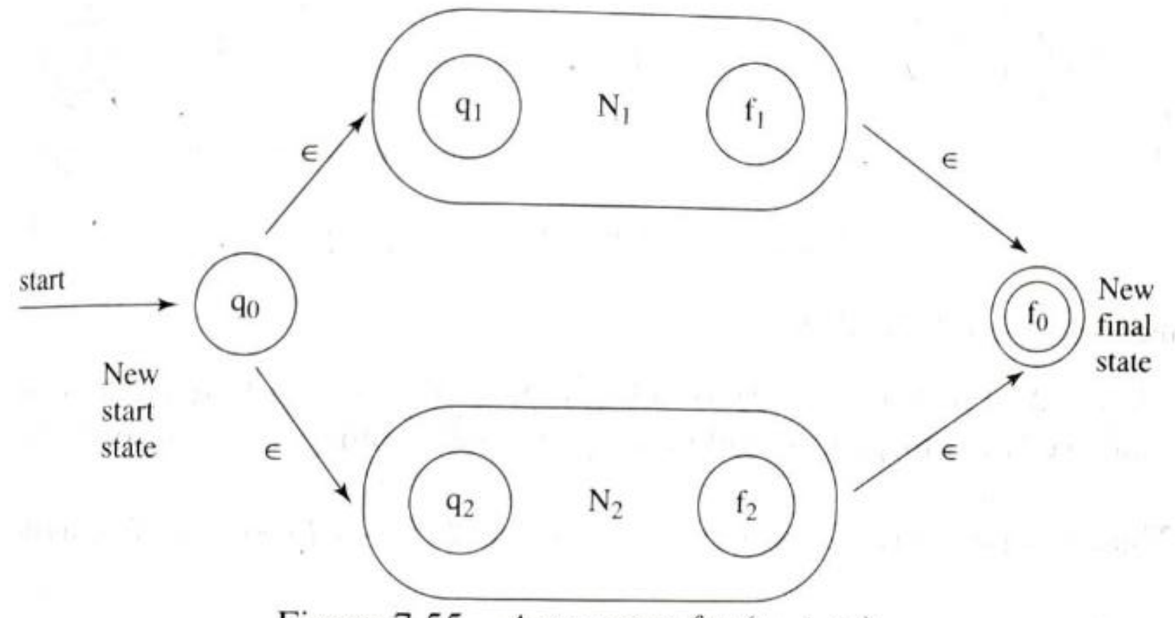


Figure 7.55. *Automaton for* $(r_1 + r_2)$.

The above diagram shows that:

- Any path of N from q_0 to f_0 must begin by either going to q_1 or q_2 on ϵ.
- If a path goes to q_1, then follow any path in N_1 to f_1 and then go to f_0 on ϵ.
- If a path goes to q_2, then follow any path in N_2 to f_2 and then go to f_0 on ϵ.

Thus, $L(N) = L(N_1) \cup L(N_2)$ as desired.

Case 2: $r = r_1 \cdot r_2$

Let N_1 and N_2 be two NFA's with ϵ-transitions accepting languages L_1 and L_2, defined by *RE*s r_1 and r_2 respectively.

$N_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\})$; $N_2 = (Q_2, \Sigma_2, \delta_2, q_2, \{f_2\})$ with $L(N_1) = L(r_1)$, $L(N_2) = L(r_2)$.

Now, with q_1 and f_2 as initial and final states, construct

$$N = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta, q_1, \{f_2\}) \text{ whose } \delta \text{ is defined by}$$

a. $\delta(q, a) = \delta_1(q, a),\ q \in Q_1 - \{f_1\},\ a \in \Sigma_1 \cup \{\epsilon\}$
b. $\delta(f_1, \epsilon) = \{q_2\}$
c. $\delta(q, a) = \delta_2(q, a),\ q \in Q_2,\ a \in \Sigma_2 \cup \{\epsilon\}$.

NFA for the above transition is given in figure 7.56.

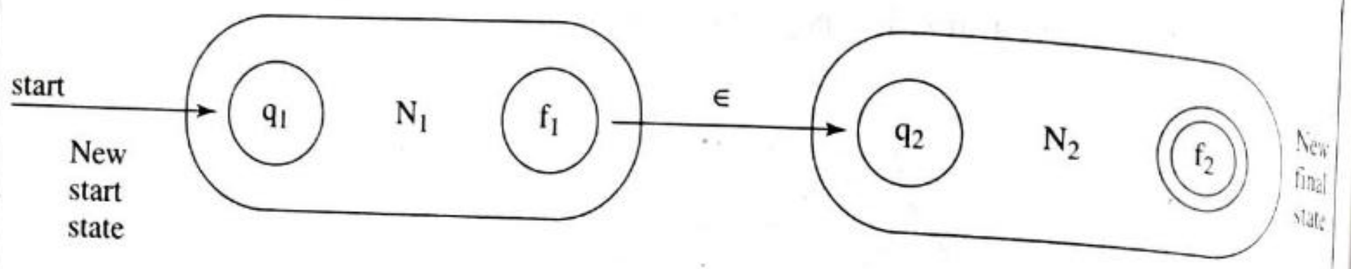


Figure 7.56. *Automaton for $r_1 \cdot r_2$.*

The above diagram shows that:

- Any path of N, from q_1 to f_2 must begin by going from q_1 to f_1 for some string x, followed by f_1 to q_2 on ϵ, followed by any path from q_2 to f_2 for some string y.

Thus, $L(N) = \{xy | x \in L(N_1),\ y \in L(N_2)\}$ and $L(N) = L(N_1) \cdot L(N_2)$, as desired.

Case 3: $r = r_1^*$

Let N_1 be with ϵ-transition and accepting the language L_1, defined by the NFA *RE* r_1.

$$N_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\}) \text{ with } L(N_1) = L(r_1).$$

Let q_0, f_0 be the initial and final states. Then, construct

$$N = (Q_1 \cup \{q_0, f_0\}, \Sigma_1, \delta, q_0, \{f_0\})$$

where δ is defined by

a. $\delta(q_0, \epsilon) = \delta(f_1, \epsilon) = \{q_1, f_0\}$.
b. $\delta(q, a) = \delta_1(q, a),\ q \in Q_1 - \{f_1\},\ a \in \Sigma_1 \cup \{\epsilon\}$.

The NFA for the above transition is given in figure 7.57.

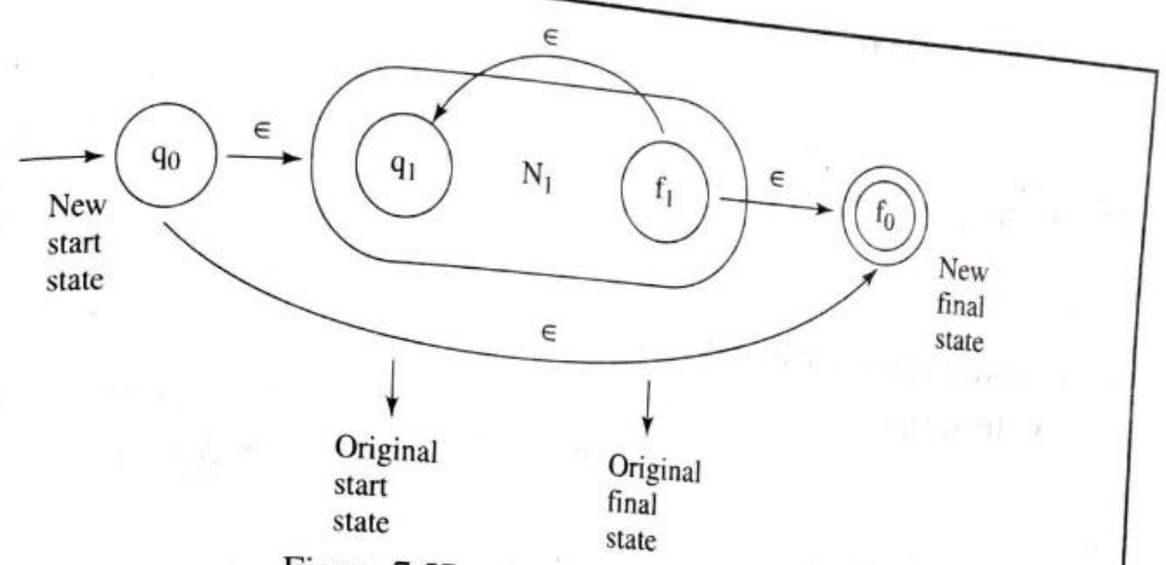


Figure 7.57. *Automaton for r_1^*.*

The above diagram shows that:

- any path from q_0 to f_0 consists either the path from q_0 to f_0 on ϵ or a path from q_0 to f_1 on ϵ, followed by some number (possibly 0) of paths from q_1 to f_1, then back to q_1 on ϵ (in L_1, labelled by a string) followed by a path from q_1 to f_1 (in a string in L), then to f_0 on ϵ.

Thus, $L(N) = L(N_1)^*$ as desired.

Hence, proved.

EXAMPLE 7.6.1: Construct an NFA with ϵ-moves for $00^* + 1$.

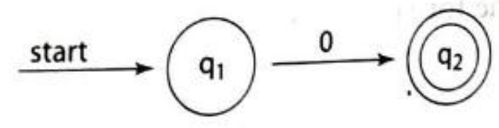
Solution:

Let $r = 00^* + 1$.

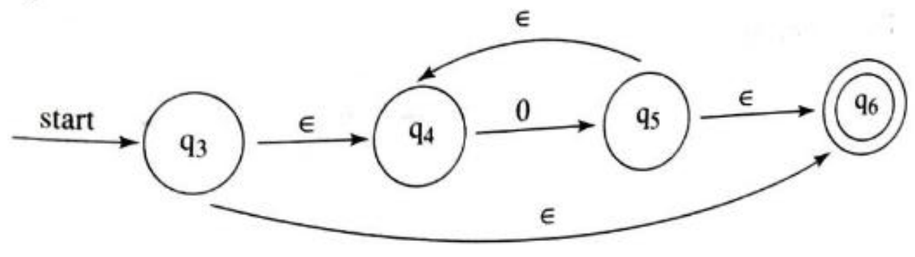
$\Rightarrow r = r_1 + r_2$ where $r_1 = 00^*,\ r_2 = 1$

a. Construct a machine for r_1:

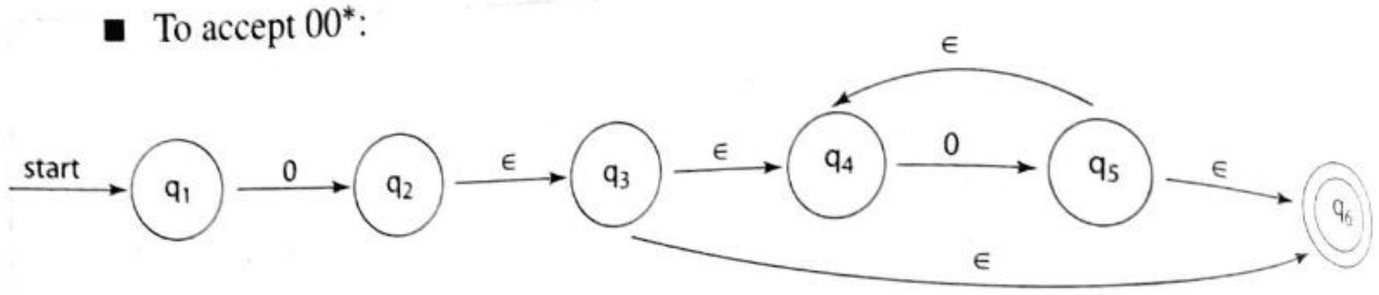
- To accept 0:



- To accept 0*:

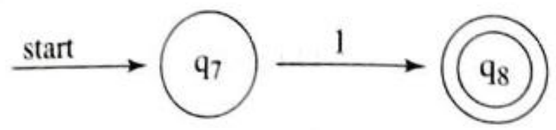


■ To accept 00^*:



b. Construct a machine for r_2:

■ To accept 1:



c. Construct a machine for r:

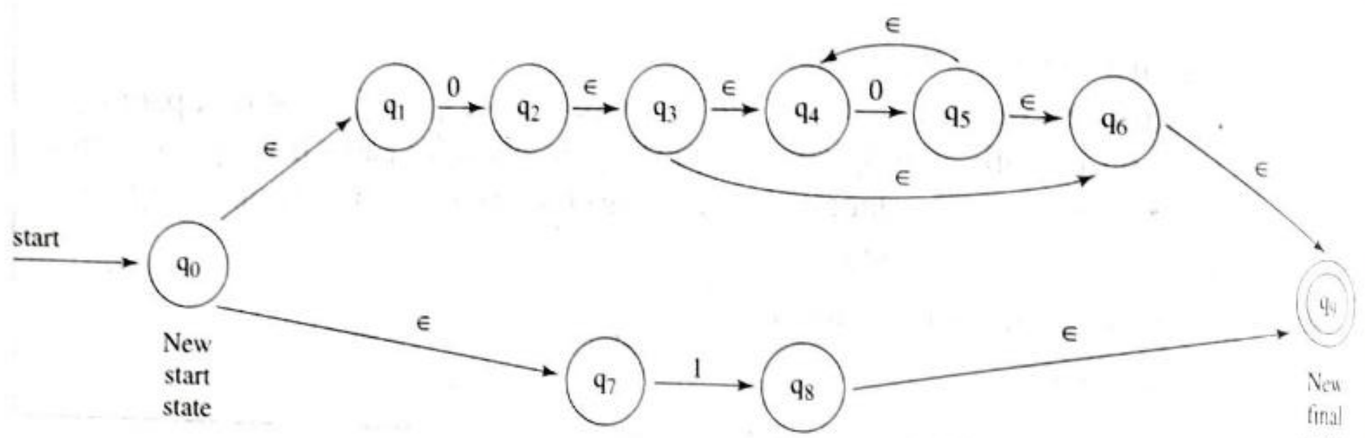


Figure 7.58. *Automaton for* $r = r_1 + r_2$

EXAMPLE 7.6.2: Construct an NFA for $r = (a + bb)^* \cdot ba^*$.

Solution:

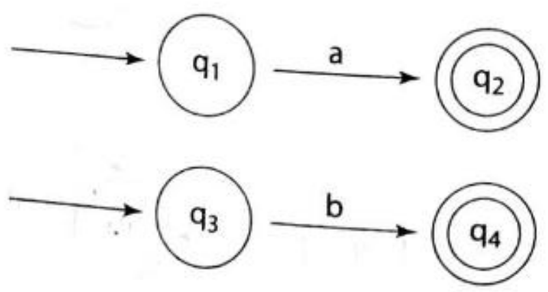
Let, $r = (a + bb)^* \cdot ba^*$.

$\Rightarrow r = r_1 \cdot r_2$ where, $r_1 = (a + bb)^*$ and $r_2 = ba^*$.

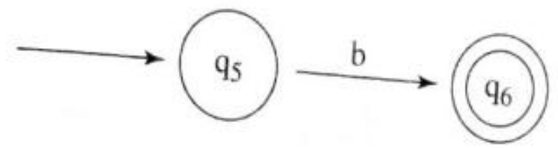
a. Construct a machine for r_1:

■ To accept a:

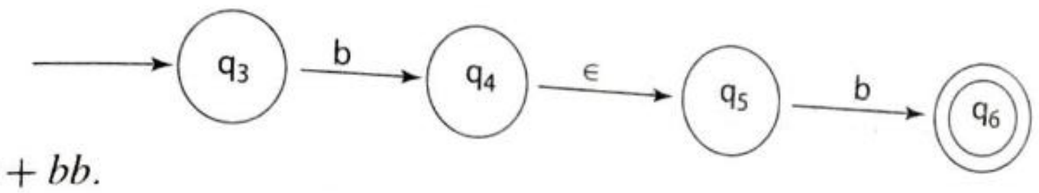
■ To accept b:



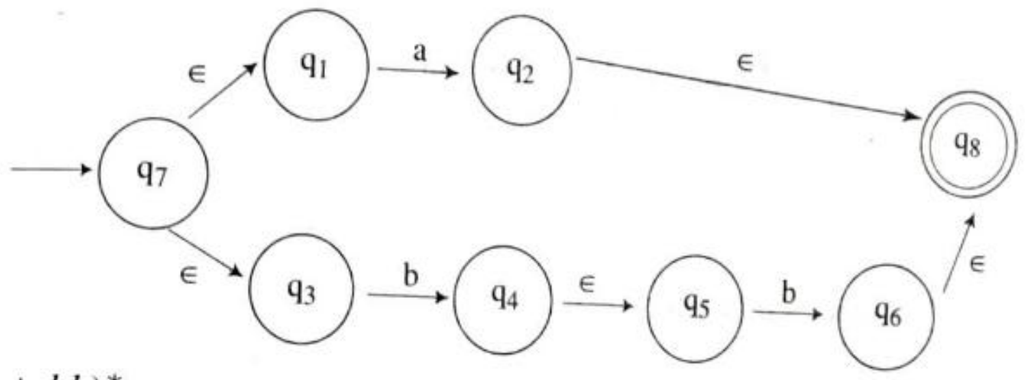
■ To accept b:



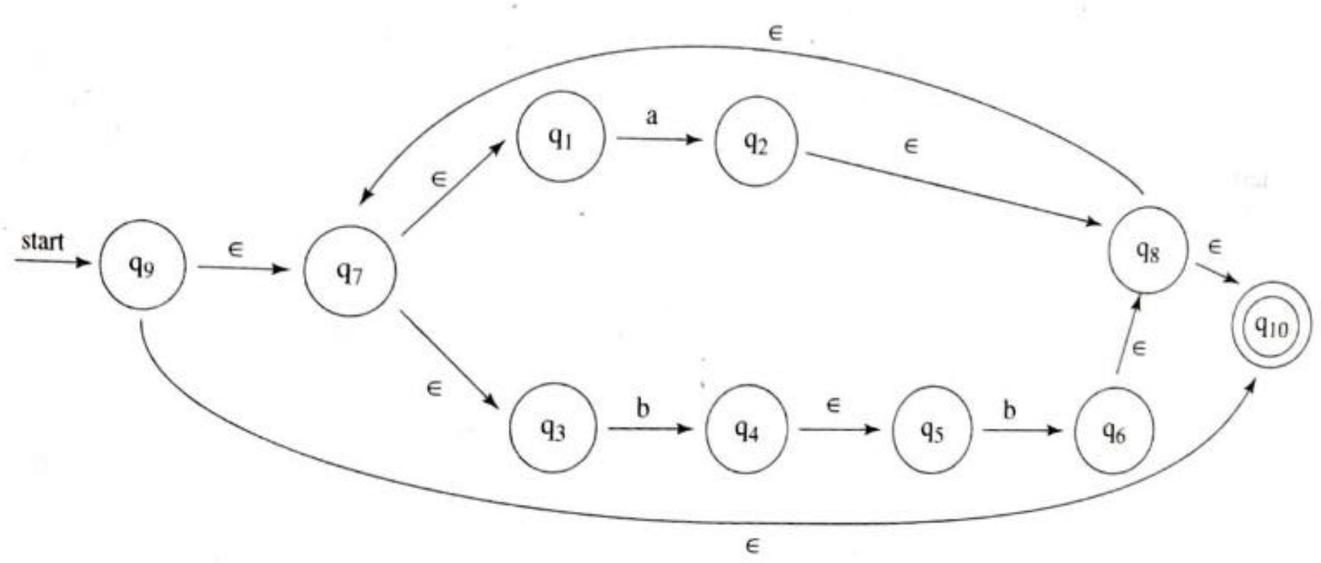
■ To accept bb:



■ To accept $a + bb$.

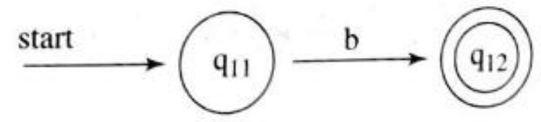


■ To accept $(a + bb)^*$.

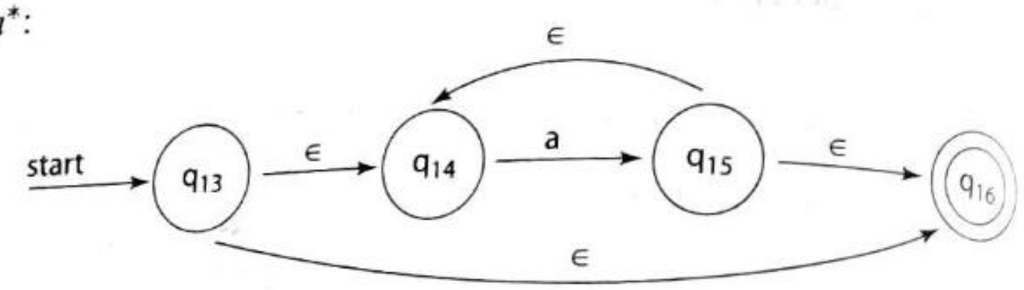


b. Construct a machine for r_2:

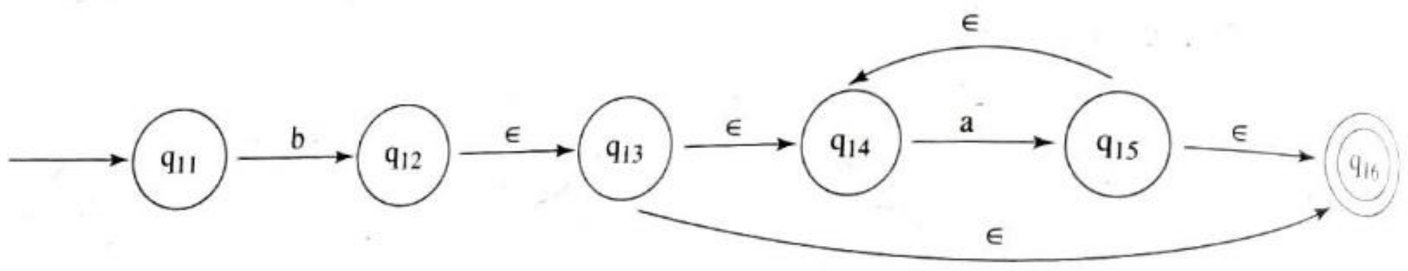
■ To accept b:



■ To accept a^*:



■ To accept ba^*:



c. Construct a machine for r:

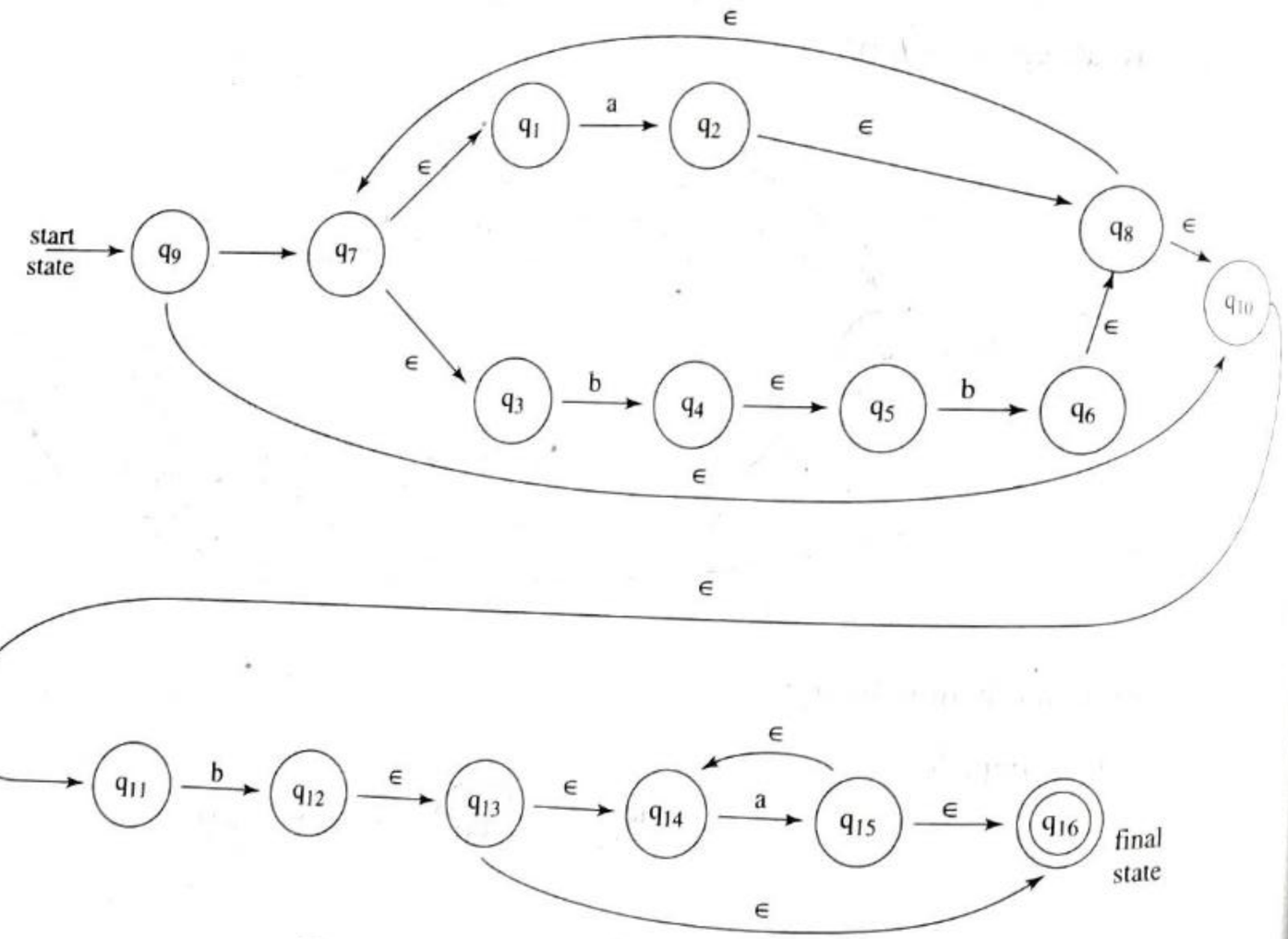


Figure 7.59. *Automaton for $r = r_1 \cdot r_2$*

EXAMPLE 7.6.3: Construct an NFA for $(0+1)^*\ 00\ (0+1)^*$.

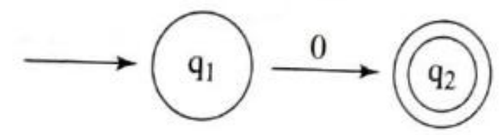
Solution:

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

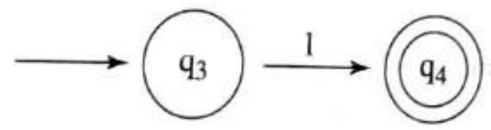
$\Rightarrow r = r_1 \cdot r_2 \cdot r_3$ where $r_1 = (0+1)^*,\ r_2 = 00,\ r_3 = (0+1)^*$.

a. Construct a machine for r_1:

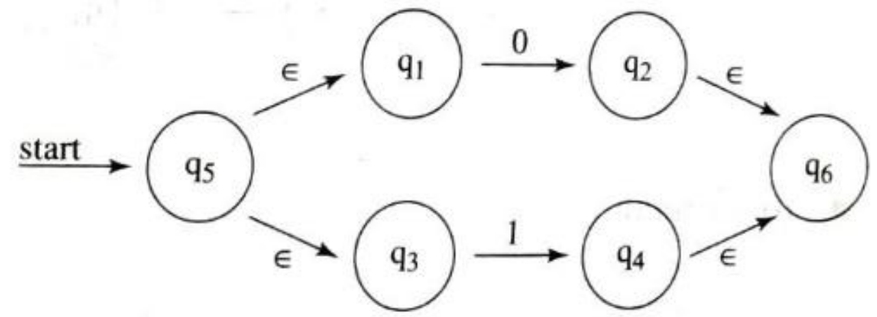
■ To accept 0:



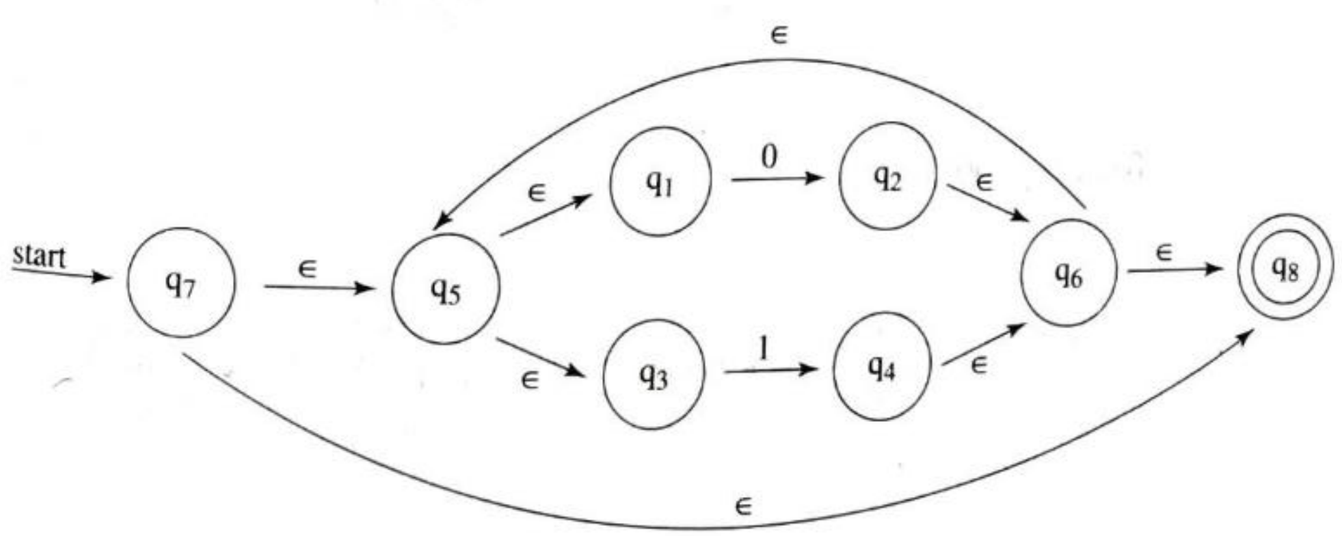
■ To accept 1:



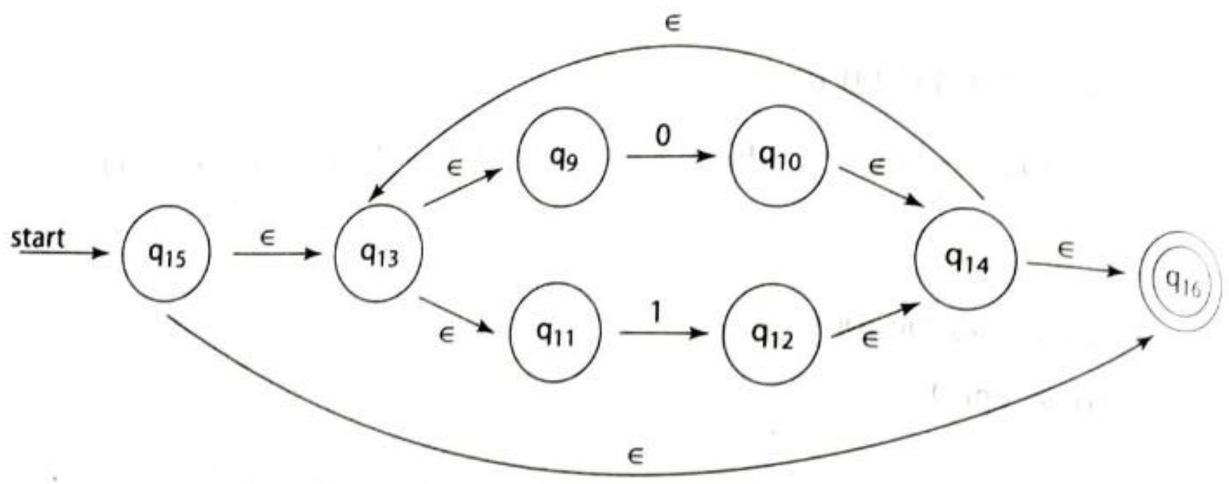
■ To accept (0+1):



■ To accept $(0+1)^*$:

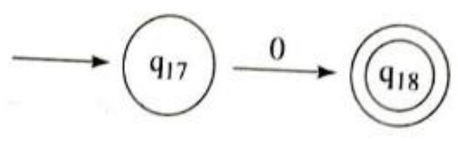


b. Construct a machine for r_3:

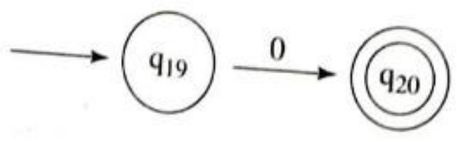


c. Construct a machine for r_2:

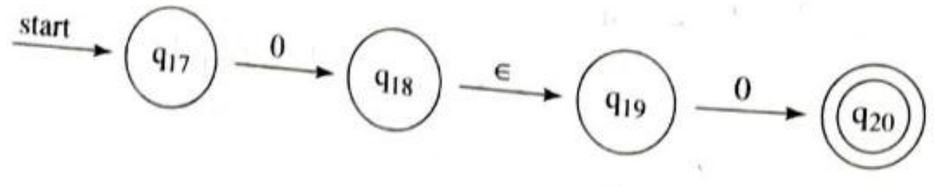
- To accept 0:



- To accept 0:



- To accept 00:



d. Construct a machine for r:

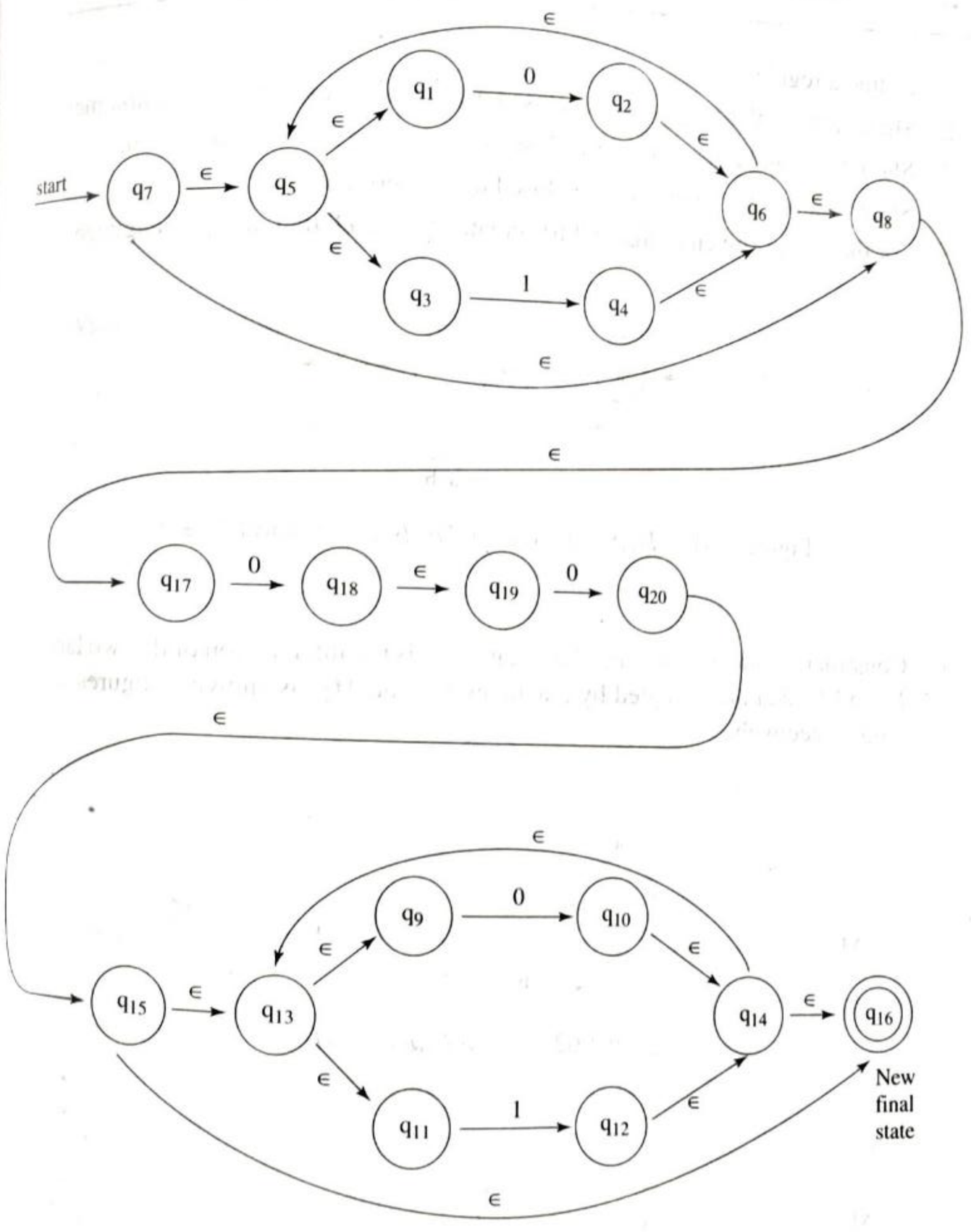


Figure 7.60. *Automaton for* $r = r_1 \cdot r_2 \cdot r_3$

7.7 Exercises

1. Define a regular set.
2. Show that regular languages are closed under kleene closure and complementation.
3. Show that regular languages are closed under union and concatenation.
4. Show that regular languages are closed under quotients.
5. For the DFA,D given in figure 7.61, create a DFA, D' that accepts language $\overline{L}$.

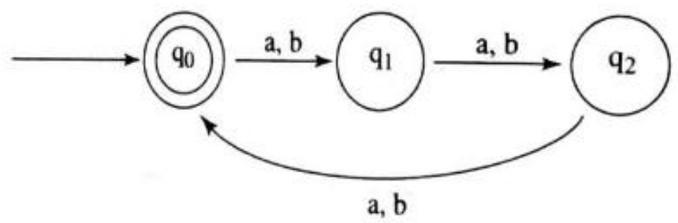


Figure 7.61. *DFA D to Accept Words of* $|w| \mod 3 = 0$.

6. Construct a finite automaton, whose language is the intersection of the two languages L_1 and L_2 that are accepted by machines M_1 and M_2, as shown in figures 7.62 and 7.63 respectively.

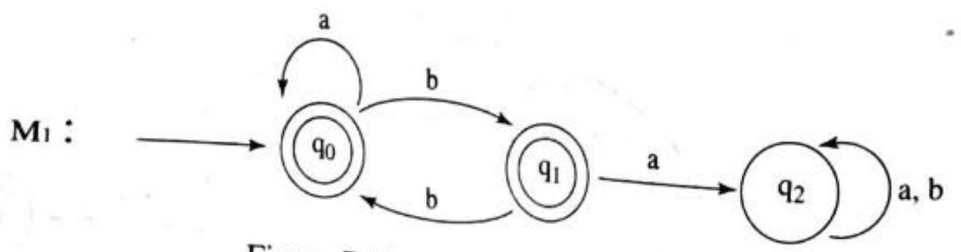


Figure 7.62. *State Diagram* M_1.

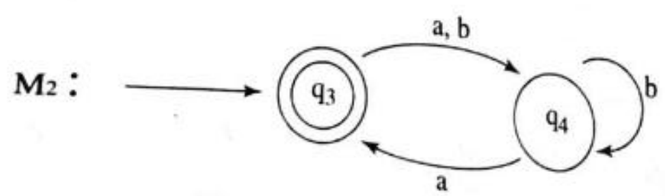


Figure 7.63. *State Diagram* M_2.

7. Obtain a regular expression for the DFA given in figure 7.64.

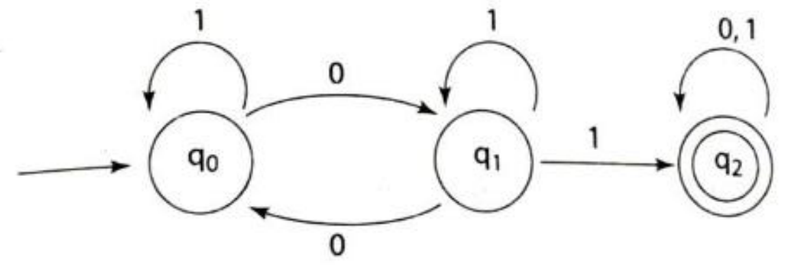


Figure 7.64. *State Diagram DFA.*

8. What is the language accepted by the following DFAs?

a.

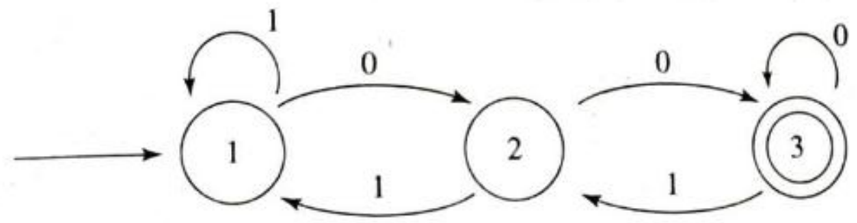


Figure 7.65. *State Diagram.*

b.

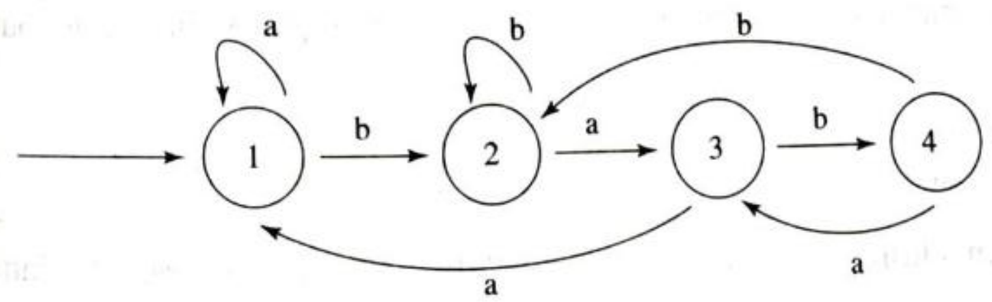


Figure 7.66. *State Diagram.*

9. Prove that, for every regular expression r, there exists an NFA with ϵ-transitions that accepts $L(r)$.
10. Construct NFA for the following regular expressions.
    a. $0^*0^* + 1^*$
    b. $0^*1^*2^*$
    c. $(0+1)^*\,00\,(0+1)$
    d. $0^* + 1^* + 2^*$
    e. $(ab^* + b)^*$
    f. $(a+b)^*\,(aa+bb)\,(a+b)^*$