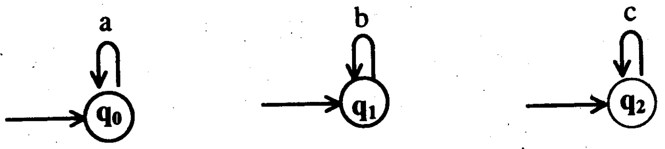
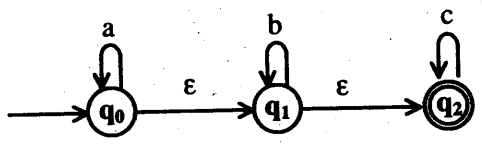
The ε-NFA that accepts strings of zero or more a's, zero or more b's and zero or more c's can be represented as shown below:



It is given that zero or more a's should be followed by zero or more b's followed by zero or more c's and the corresponding ε-NFA is shown below:



Note: The equivalent DFA can be obtained as shown in example 2.21

2.17 Conversion from ε-NFA to DFA

We have seen in the previous section that an NFA can be converted into DFA using subset construction. On similar lines, it is possible to convert an ε-NFA to DFA and the procedure is shown below:

Let $M_E = (Q_E, \Sigma, \delta_E, q_{0E}, F_E)$ be an ε-NFA where Q_E is set of finite states, Σ is set of input alphabets (from which a string can be formed), δ_E is transition from Q x $\{\Sigma \cup \varepsilon\}$ to 2^Q, q_{0E} is the start state and F_E is the set of final or accepting states. The equivalent DFA

$$M_D = (Q_D, \Sigma, \delta_D, q_{0D}, F_D)$$

can be obtained as shown below:

Step1:
q_{0D} = ε-CLOSURE(q_{0E}). This indicates that the start state of DFA can be obtained by taking the ε-CLOSURE of the start state of ε-NFA.

Step2:
A state of DFA represented by $[p_1, p_2, \ldots p_k] \in Q_D$ can be obtained from ε-NFA using the following transition:

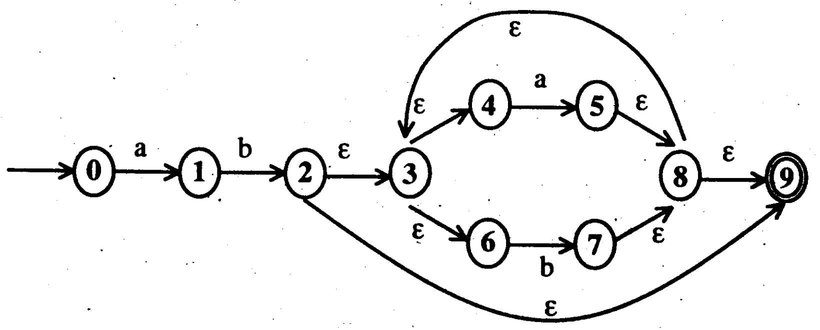
$$\delta_D([p_1, p_2, \ldots p_k], a) = \varepsilon\text{-CLOSURE}(\ \delta_N(p_1,a) \cup \delta_N(p_2,a) \cup \ldots\ldots \delta_N(p_k,a)\ )$$

$$= \bigcup_{i=1}^{k} \varepsilon\text{-CLOSURE}(\delta_N(p_i,a)) \text{ for each } a \in \Sigma.$$

Step 3:

If $[p_1, p_2, \ldots p_k] \in Q_D$ and if any of the state $p_i \in F_E$ (set of accepting states), then $[p_1, p_2, \ldots p_k]$ is a final state in DFA.

Example 2.29: Convert the following NFA to its equivalent DFA.



The start state of DFA i.e., $q_{0D} = \varepsilon\text{-CLOSURE}(0) = \{0\}$ (A)

Consider the state [A]:

When input is *a*:

$\delta(A, a) = \varepsilon\text{-CLOSURE}(\delta_N(A, a))$
$= \varepsilon\text{-CLOSURE}(\delta_N(0, a))$
$= \{1\}$ (B)

When input is *b*:

$\delta(A, b) = \varepsilon\text{-CLOSURE}(\delta_N(A, b))$
$= \varepsilon\text{-CLOSURE}(\delta_N(0, b))$
$= \{\phi\}$

Consider the state [B]:

When input is *a*:

$\delta(B, a) = \varepsilon\text{-CLOSURE}(\delta_N(B, a))$
$= \varepsilon\text{-CLOSURE}(\delta_N(1, a))$
$= \{\phi\}$

When input is *b*:

$\delta(B, b) = \varepsilon\text{-CLOSURE}(\delta_N(B, b))$
$= \varepsilon\text{-CLOSURE}(\delta_N(1, b))$
$= \varepsilon\text{-CLOSURE}(\{2\})$
$= \{2,3,4,6,9\}$ (C)

This is because, in state 2, due to ε-transitions (or without giving any input) there can be transition to states 2,3,4,6,9 also. So, all these states are reachable from state 2. Therefore,

$\delta(B, b) = \{2,3,4,6,9\} = C$

Consider the state [C]:

When input is *a*:

$\delta(C, a)$ = ε-CLOSURE $(\delta_N(C, a))$
= ε-CLOSURE$(\delta_N(\{2,3,4,6,9\}, a))$
= ε-CLOSURE $\{5\}$
= $\{5, 8, 9, 3, 4, 6\}$
= $\{3, 4, 5, 6, 8, 9\}$ (ascending order) (D)

This is because, in state 5 due to ε-transitions, the states reachable are $\{5, 8, 9, 3, 4, 6\}$. Therefore,

$\delta(C, a) = \{3, 4, 5, 6, 8, 9\} = D$

When input is *b*:

$\delta(C, b)$ = ε-CLOSURE $(\delta_N(C, b))$
= ε-CLOSURE$(\delta_N(\{2, 3, 4, 6, 9\}, b)$
= ε-CLOSURE $(\{7\})$
= $\{7, 8, 9, 3, 4, 6\}$
= $\{3, 4, 6, 7, 8, 9\}$(ascending order) (E)

This is because, from state 7 the states that are reachable without any input (i.e., ε-transition) are $\{7, 8, 9, 3, 4, 6\}$. Therefore,

$\delta(C, b) = \{3, 4, 6, 7, 8, 9\} = E$

Consider the state [D]:

When input is *a*:

$\delta(D, a)$ = ε-CLOSURE$(\delta(D, a))$
= ε-CLOSURE $(\delta_N(\{3,4,5,6,8,9\}, a))$
= ε-CLOSURE $(\{5\})$
= $\{5, 8, 9, 3, 4, 6\}$
= $\{3, 4, 5, 6, 8, 9\}$ (ascending order) (D)

When input is *b*:

$\delta(D, b)$ = ε-CLOSURE$(\delta(D, b))$
= ε-CLOSURE $(\delta_N(\{3,4,5,6,8,9\}, b))$
= ε-CLOSURE $(\{7\})$
= $\{7, 8, 9, 3, 4, 6\}$
= $\{3, 4, 6, 7, 8, 9\}$ (ascending order) (E)

Consider the state [E]:

When input is *a*:

$\delta(E, a)$ = ε-CLOSURE(δ(E, a))
= ε-CLOSURE (δ_N({3,4,6,7,8,9}, a))
= ε-CLOSURE ({5})
= {5, 8, 9, 3, 4, 6}
= {3, 4, 5, 6, 8, 9}(ascending order)
(D)

When input is *b*:

$\delta(E, b)$ = ε-CLOSURE(δ(E, b))
= ε-CLOSURE (δ_N({3,4,6,7,8,9}, b))
= ε-CLOSURE ({7})
= {7, 8, 9, 3, 4, 6}
= {3, 4, 6, 7, 8, 9}(ascending order)
(E)

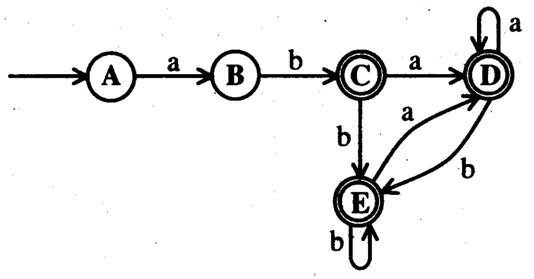
Since there are no new states, we can stop at this point and the transition table for the DFA is shown in table 2.16.

$\longleftarrow \Sigma \longrightarrow$

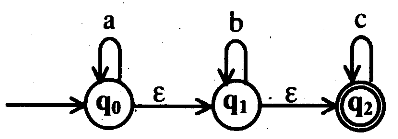
	δ	a	b
	A	B	-
	B	-	C
Q	C	D	E
	D	D	E
	E	D	E

Table 2.16 Transitional table

The states C,D and E are final states, since 9 (final state of NFA) is present in C, D and E. The final transition diagram of DFA is shown in figure 2.37.



Example 2.30: Convert the following NFA to DFA



The state q_0 is the start of ε-NFA and so the start state of DFA is ε-CLOSURE (q_0) i.e.,

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

Now, we find the transitions from the state $\{q_0, q_1, q_2\}$ on $\Sigma = \{a, b, c\}$.

Consider the state $[q_0, q_1, q_2]$:

On input *a*:

$\delta(\{q_0, q_1, q_2\}, a) =$ ε-CLOSURE $(\{q_0\})$
$= \{q_0, q_1, q_2\}$ (A)

On input *b*:

$\delta(\{q_0, q_1, q_2\}, b) =$ ε-CLOSURE $(\{q_1\})$.
$= \{q_1, q_2\}$ (B)

On input *c*:

$\delta(\{q_0, q_1, q_2\}, c) =$ ε-CLOSURE $(\{q_2\})$
$= q_2$ (C)

The new states reachable from $\{q_0, q_1, q_2\}$ are $\{q_1, q_2\}$ and $\{q_2\}$. The transitions from these are shown below:

Consider the state $[q_1, q_2]$:

On input *a*:

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

On input *b*:

$\delta(\{q_1, q_2\}, b) =$ ε-CLOSURE $(\{q_1\})$
$= \{q_1, q_2\}$ (B)

On input *c*:

$\delta(\{q_1, q_2\}, c) =$ ε-CLOSURE $(\{q_2\})$
$= q_2$ (C)

The transitions from the state q_2 are shown below:

Consider the state $[q_2]$:

On input *a*:

$\delta(q_0, a) = \phi$

On input *b*:

$\delta(q_1, b) = \phi$

On input *c*:

$\delta(q_2, c) = \varepsilon\text{-CLOSURE}(q_2)$

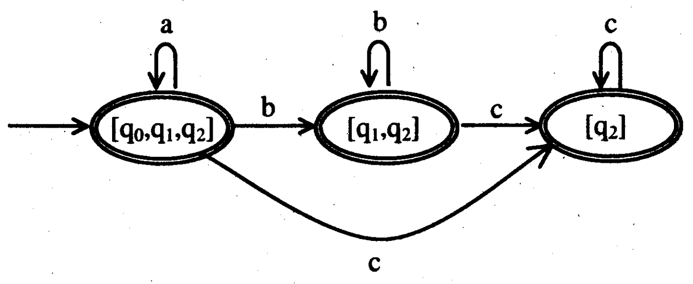
$= q_2$ (C)

The transition table along with transition diagram is shown below:

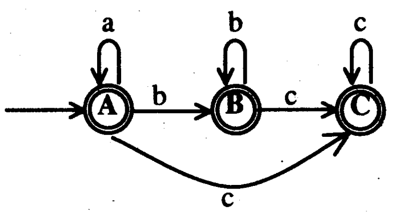
$\longleftarrow \quad \Sigma \quad \longrightarrow$

Q

δ	a	b	c
$[q_0,q_1,q_2]$	$[q_0, q_1, q_2]$	$[q_1, q_2]$	$[q_2]$
$[q_1,q_2]$	ϕ	$[q_1,q_2]$	$[q_2]$
$[q_2]$	ϕ	ϕ	$[q_2]$



The DFA can also be written as shown below:



where A = $[q_0,q_1,q_2]$, B = $[q_1,q_2]$ and C = $[q_2]$

Example 2.31: Obtain an NFA with ε-transitions (ε-NFA) to accept decimal numbers and then obtain the equivalent DFA.

Note: A decimal number has

1. an optional + or – sign followed by a '.' Followed by string of digits

   or

2. an optional + or – sign followed by a string of digits followed by a '.' and in turn followed by string of digits

The ε-NFA is shown in figure 2.39.

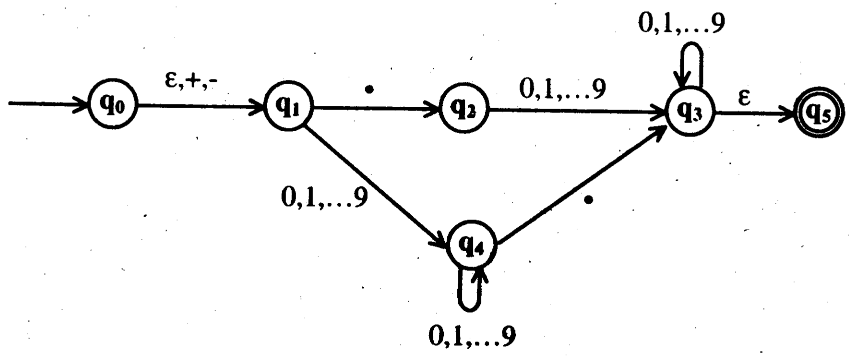


Fig. 2.39 ε-NFA to accept decimal number

Here, the machine M = (Q, Σ, δ, q_0, F) where

Q = {q_0, q_1, q_2, q_3, q_4, q_5}

Σ = { +, -, ., 0, 1, 2,....9}

q_0 is the start state

F = { q_5 } is the final state

δ is shown using the transition table 2.18.

The DFA for the above machine can be obtained by modifying the subset construction for ε-transition. The start state of DFA is the ε-CLOUSRE of start state of ε-NFA.

i.e., q_{0D} = ε-CLOSURE(q_0) = {q_0,q_1} (A)

← Σ →

δ	ε	+	-	.	0, 1,...9
q_0	q_1	q_1	q_1	φ	φ
q_1	φ	φ	φ	q_2	q_4
q_2	φ	φ	φ	φ	q_3
q_3	q_5	φ	φ	φ	q_3
q_4	φ	φ	φ	q_3	q_4
q_5	φ	φ	φ	φ	φ

↑ Q ↓

Consider the state [A]:

When input is ±:

$\delta(A, \pm) = \varepsilon\text{-CLOSURE}(\delta_N(A, \pm))$
$= \varepsilon\text{-CLOSURE}(\delta_N(\{q_0, q_1\}, \pm))$
$= \varepsilon\text{-CLOSURE}(q_1)$
$= \{q_1\}$ (B)

When input is .

$\delta(A, .) = \varepsilon\text{-CLOSURE}(\delta_N(A, .))$
$= \varepsilon\text{-CLOSURE}(\delta_N(\{q_0, q_1\}, .))$
$= \varepsilon\text{-CLOSURE}(q_2)$
$= \{q_2\}$ (C)

When input is 0,1,2,...9

$\delta(A, \{0,1,\ldots 9\}) = \varepsilon\text{-CLOSURE}(\delta_N(A, \{0,1,\ldots 9\}))$
$= \varepsilon\text{-CLOSURE}(\delta_N(\{q_0, q_1\}, \{0,1,\ldots 9\}))$
$= \varepsilon\text{-CLOSURE}(q_4)$
$= \{q_4\}$ (D)

Consider the state [B]:

When input is ±:

$\delta(B, \pm) = \varepsilon\text{-CLOSURE}(\delta_N(B, \pm))$
$= \varepsilon\text{-CLOSURE}(\delta_N(q_1, \pm))$
$= \varepsilon\text{-CLOSURE}(\phi)$
$= \phi$

When input is .

$\delta(B, .) = \varepsilon\text{-CLOSURE}(\delta_N(B, .))$
$= \varepsilon\text{-CLOSURE}(\delta_N(q_1, .))$
$= \varepsilon\text{-CLOSURE}(q_2)$
$= \{q_2\}$ (C)

When input is 0,1,2,...9

$\delta(B, \{0,1,\ldots 9\}) = \varepsilon\text{-CLOSURE}(\delta_N(B, \{0,1,\ldots 9\}))$
$= \varepsilon\text{-CLOSURE}(\delta_N(q_1, \{0,1,\ldots 9\}))$
$= \varepsilon\text{-CLOSURE}(q_4)$
$= \{q_4\}$ (D)

Consider the state [C]:

When input is ±:

$\delta(C, \pm)$ = ε-CLOSURE ($\delta_N(C, \pm)$)
= ε-CLOSURE ($\delta_N(q_2, \pm)$)
= ε-CLOSURE (ϕ)
= ϕ

When input is .

$\delta(C, .)$ = ε-CLOSURE ($\delta_N(C, .)$)
= ε-CLOSURE ($\delta_N(q_2, .)$)
= ε-CLOSURE (ϕ)
= ϕ

When input is 0,1,2,...9

$\delta(C, \{0,1,\ldots 9\})$ = ε-CLOSURE ($\delta_N(C, \{0,1,\ldots 9\})$)
= ε-CLOSURE ($\delta_N(q_2, \{0,1,\ldots 9\})$)
= ε-CLOSURE (q_3)
= $\{q_3, q_5\}$
(E)

Consider the state [D]:

When input is ±:

$\delta(D, \pm)$ = ε-CLOSURE ($\delta_N(D, \pm)$)
= ε-CLOSURE ($\delta_N(q_4, \pm)$)
= ε-CLOSURE (ϕ)
= ϕ

When input is .

$\delta(D, .)$ = ε-CLOSURE ($\delta_N(D, .)$)
= ε-CLOSURE ($\delta_N(q_4, .)$)
= ε-CLOSURE (q_3)
= $\{q_3, q_5\}$
(E)

When input is 0,1,2,...9

$\delta(D, \{0,1,\ldots 9\})$ = ε-CLOSURE ($\delta_N(D, \{0,1,\ldots 9\})$)
= ε-CLOSURE ($\delta_N(q_4, \{0,1,\ldots 9\})$)
= ε-CLOSURE (q_4)
= $\{q_4\}$ (D)

Consider the state [E]:

When input is ±:

$$\begin{aligned}\delta(E, \pm) &= \varepsilon\text{-CLOSURE}(\delta_N(E, \pm))\\ &= \varepsilon\text{-CLOSURE}(\delta_N(\{q_3, q_5\}, \pm))\\ &= \varepsilon\text{-CLOSURE}(\phi)\\ &= \phi\end{aligned}$$

When input is .

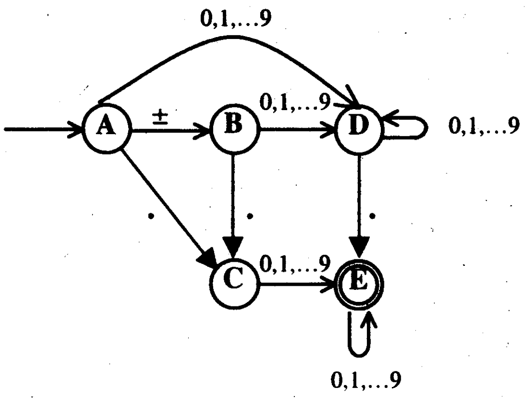
$$\begin{aligned}\delta(E, .) &= \varepsilon\text{-CLOSURE}(\delta_N(E, .))\\ &= \varepsilon\text{-CLOSURE}(\delta_N(\{q_3, q_5\}, .))\\ &= \varepsilon\text{-CLOSURE}(\phi)\\ &= \phi\end{aligned}$$

When input is 0,1,2,...9

$$\begin{aligned}\delta(E, \{0,1,\ldots 9\}) &= \varepsilon\text{-CLOSURE}(\delta_N(E, \{0,1,\ldots 9\}))\\ &= \varepsilon\text{-CLOSURE}(\delta_N(\{q_3, q_5\}, \{0,1,\ldots 9\}))\\ &= \varepsilon\text{-CLOSURE}(q_3)\\ &= \{q_3, q_5\}\\ &\quad (E)\end{aligned}$$

The transition table along with transition diagram is shown below:

δ	+	-	.	0, 1,...9
A	B	B	C	D
B	ϕ	ϕ	C	D
C	ϕ	ϕ	ϕ	E
D	ϕ	ϕ	E	D
*E	ϕ	ϕ	ϕ	E



In this DFA $M = (Q, \Sigma, \delta, q_0, F)$ where

$Q = \{A, B, C, D, E\}$

$\Sigma = \{ +, -, ., 0, 1, 2, \ldots 9\}$

A is the start state

$F = \{ E \}$ is the final state

δ is shown sing the transition table 2.19.

Note: Since $E = \{q_3, q_5\}$ has a final state of NFA, E is the final state in DFA. The corresponding DFA is shown using the transition diagram in figure 2.40.

2.18 Difference between DFA, NFA and ε-NFA

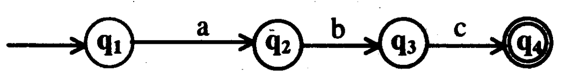
Now, let us see **"What is the difference between DFA, NFA and ε-NFA?"** Strictly speaking the difference between DFA and NFA lies only in the definition of δ. Using this difference some more points can be derived and can be written as shown:

DFA	NFA	ε-NFA
The DFA is a 5-tuple or quintuple $M = (Q, \Sigma, \delta, q_0, F)$ where Q is set of finite states Σ is set of input alphabets $\delta : Q \times \Sigma$ to Q q_0 is the start state $F \subseteq Q$ is set of final states	An NFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where Q is set of finite states Σ is set of input alphabets $\delta : Q \times \Sigma$ to subsets of 2^Q q_0 is the start state $F \subseteq Q$ is set of final states	An ε-NFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where Q is set of finite states Σ is set of input alphabets $\delta : Q \times (\Sigma \cup \varepsilon)$ to subsets of 2^Q q_0 is the start state $F \subseteq Q$ is set of final states
If a string *w* is accepted by DFA from state *q*, there will be exactly one path labeled *w* starting at *q*. Therefore, to determine if a string is accepted, it is sufficient to check for this one path. Following this path, if the machine is in final state, we say that the string *w* is accepted. Otherwise, the string is rejected.	If a string *w* is accepted by an NFA from state *q*, there may be many paths labeled *w*. Therefore, to determine if a string is accepted, it is necessary to check all these paths to see whether one or more terminate at final state. Following these paths, if we reach a final state, the string is accepted else the string is rejected.	For an ε-NFA, there may be many paths labeled *w* along with ε and all must be checked to see whether one or more terminate at final state. If final state is reached, the given string *w* is accepted by the machine else it is rejected by the machine.

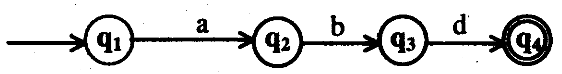
There can be zero or one transition from a state on an input symbol	There can be zero, one or more transitions from a state on an input symbol	There can be zero, one or more transitions from a state with or without giving any input
Difficult to construct	Easy to construct	Easy to construct using regular expressions
Figure 2.1	Figure 2.34	Figure 2.36

Example 2.32: Obtain a NFA's to recognize the strings *abc*, *abd* and aacd assuming $\Sigma = \{a, b, c, d\}$

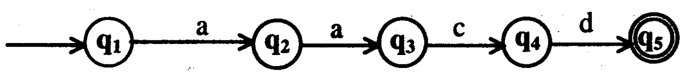
The machine to accept the string abc is shown below:



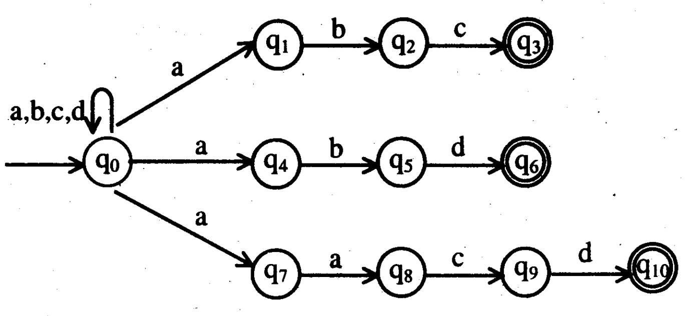
The machine to accept the string abd is shown below:



The machine to accept the string aacd is shown below:

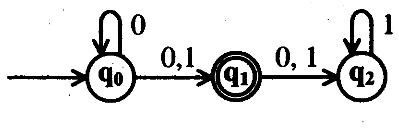


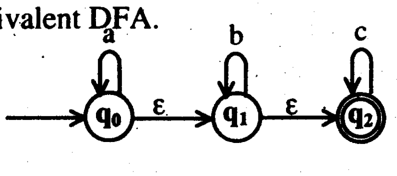
But, all these strings can be preceded by strings of a's, b's, c's and d's and observe that the first symbol in each of these machines is *a*. So, the complete NFA to accept the strings abc, abd and aacd can be written as shown below:

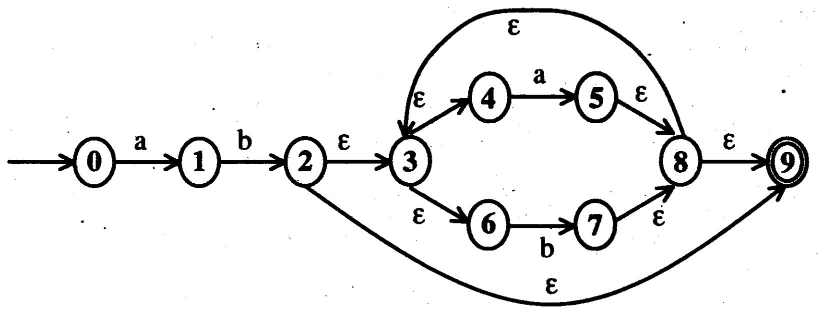


Exercises:

1. What is an NFA? Explain with example
2. What is the need for an NFA?
3. What is the difference DFA and NFA?
4. Give a general procedure to convert an NFA to DFA.
5. Convert the following NFAs into an equivalent DFA.







6. Define distinguishable and non-distinguishable states
7. Give the general procedure to minimize the states of DFA
8. Construct a DFA which accepts strings of 0's and 1's where the value of each string is represented as a binary number. Only the strings representing zero modulo five should be accepted. For example, 0000,0101,1010,1111 etc. should be accepted. After constructing the DFA, obtain the minimum DFA.
9. Obtain a DFA to accept string of 0's and 1's having three consecutive zeros.
   Ans: $\delta(q_0,0) = q_1$, $\delta(q_0,1) = q_0$, $\delta(q_1,0) = q_2$, $\delta(q_1,1) = q_0$,
   $\delta(q_2,0) = q_3$, $\delta(q_2,1) = q_0$, $\delta(q_3,0) = q_3$, $\delta(q_3,1) = q_3$
   q_0 is the start state and q_3 is the final state.

12. Draw a DFA which accepts strings of 0's and 1's ending with 1 and does not contain a substring 00.

Ans: $\delta(q_0,0) = q_1$, $\delta(q_0,1) = q_2$, $\delta(q_1,0) = q_3$, $\delta(q_1,1) = q_2$,
$\delta(q_2,0) = q_1$, $\delta(q_2,1) = q_2$, $\delta(q_3,0) = q_3$, $\delta(q_3,1) = q_3$
q_0 is the start state and q_2 is the final state.
q_3 is the trap state or dead state

13. Draw a DFA to accept strings of 0's and 1's ending with the string 10.
Ans: $\delta(q_0,0) = q_0$, $\delta(q_0,1) = q_1$, $\delta(q_1,0) = q_2$, $\delta(q_1,1) = q_1$,
$\delta(q_2,0) = q_0$, $\delta(q_2,1) = q_1$ where q_2 is the final state.

14. Draw a DFA to accept the language L = { (00)*(11)*}
Ans: $\delta(q_0,0) = q_1$, $\delta(q_0,1) = q_2$, $\delta(q_1,0) = q_0$, $\delta(q_1,1) = q_4$, $\delta(q_2,0) = q_4$, $\delta(q_2,1) = q_3$, $\delta(q_3,0) = q_4$, $\delta(q_3,1) = q_2$, $\delta(q_4,0) = q_4$, $\delta(q_4,1) = q_4$
q_0 is the start state, q_0 and q_3 are the final states, q_4 is the trap state or dead state

15. Draw an NFA to accept the string of a's and b's such that it can accept either the string consisting of one a followed by any number of a's or one b followed by any number of b's (i.e., aa* | bb*) and obtain the corresponding DFA

3 Regular Expressions and Languages

What we will know after reading this chapter?

- Definition of Regular expressions and method of representing
- Meaning of various regular expressions
- To obtain regular expressions for given languages
- Relation between FA and Regular Expression
- To obtain NFA from regular expression along with theorem
- Solution to obtain NFAs for various types of regular expressions
- To obtain regular expressions from various types of FAs
- Applications of Regular Expressions
- Chomsky hierarchy- definition of type-0 (un-restricted grammar), type-1 (context sensitive grammar), type-2 (Context free grammar), type-3 (Regular grammar/ linear grammar)
- To obtain FA from regular grammar or right-linear grammar along with the proof
- Solution to obtain FA from varieties of regular grammars
- To obtain regular grammar from FA along with various types of problems and solutions
- To obtain left linear grammar from FA along with various types of problems and solutions
- Solution to more than 35 problems of various nature

In chapter 2, we saw that any language that can be accepted by a finite automaton is called regular language i.e., we can construct either DFA or NFA to recognize the regular language. This chapter covers regular expressions which is another way to express the regular languages. The way to obtain equivalent NFAs for the regular expressions is also discussed. It also covers regular grammar from which a regular language can be obtained and method of obtaining finite automaton from the grammar.

3.1 Regular Expression

A regular language can be described using regular expressions in the form of notations consisting of the symbols such as alphabets in Σ, the operators such as '.', '+' and '*'. The three basic operations used to obtain a regular expression are union, closure operation and concatenation.

The operator + is used for union, the symbol * is used for closure operation and the symbol '.' is used for concatenation. If the regular expression is too complex one can use the braces '(' and ')' conveniently to understand the regular expressions. Now, let us see **"What is a regular expression?"** A regular expression can be formally defined as follows.

Definition: A regular expression is recursively defined as follows.

1. ϕ is a regular expression denoting an empty language.
2. ε-(epsilon) is a regular expression indicates the language containing an empty string.
3. *a* is a regular expression which indicates the language containing only {a}
4. If R is a regular expression denoting the language L_R and S is a regular expression denoting the language L_S, then
   a. R+S is a regular expression corresponding to the language $L_R \cup L_S$.
   b. R.S is a regular expression corresponding to the language $L_R.L_S$..
   c. R* is a regular expression corresponding to the language L_R^*.
5. The expressions obtained by applying any of the rules from 1 to 4 are regular expressions.

Note: The order of evaluation of the regular expression is determined by the parenthesis as is used in arithmetic expressions and the priority associated with other operators. Closure operator (denoted by * also called Kleen closure) is having the highest priority, then concatenation and finally the union operator.

The table 3.1 shows some examples of regular expressions and the language corresponding to these regular expressions.

Regular expressions	Meaning
(a+b)*	Set of strings of a's and b's of any length including the NULL string.
(a+b)*abb	Set of strings of a's and b's ending with the string abb
ab(a+b)*	Set of strings of a's and b's starting with the string ab.
(a+b)*aa(a+b)*	Set of strings of a's and b's having a sub string aa.
a*b*c*	Set of string consisting of any number of a's(may be empty string also)

	followed by any number of b's(may include empty string) followed by any number of c's(may include empty string).
$a^+b^+c^+$	Set of string consisting of at least one 'a' followed by string consisting of at least one 'b' followed by string consisting of at least one 'c'.
aa*bb*cc*	Set of strings consisting of at least one 'a' followed by string consisting of at least one 'b' followed by string consisting of at least one 'c'.
(a+b)* (a + bb)	Set of strings of a's and b's ending with either *a* or *bb*
(aa)*(bb)*b	Set of strings consisting of even number of a's followed by odd number of b's
(0+1)*000	Set of strings of 0's and 1's ending with three consecutive zeros(or ending with 000)
(11)*	Set of strings consisting of even number of 1's
01* + 1	The language represented is the string 1 plus the string consisting of a zero followed by any number of 1's possibly including none.
(01)* + 1	The language consists of a string 1 or strings of (01)'s that repeat zero or more times.
0(1* + 1)	Set of strings consisting of a zero followed by any number of 1's
(1+ε)(00*1)*0*	Strings of 0's and 1's without any consecutive 1's
(0+10)*1*	Strings of 0's and 1's ending with any number of 1's (possibly none)

Table 3.1 Meaning of regular expressions

Example 3.1: Obtain a regular expression to accept a language consisting of strings of a's and b's with alternate a's and b's

The alternate a's and b's can be obtained by "concatenating the string ab zero or more times" which can be represented by the regular expression

(ab)*

and adding an optional *b* to the front and adding an optional *a* at the end as shown below:

(ε + b) (ab)* (ε + a)

Thus, the complete expression is given by

(ε + b) (ab)* (ε + a)

Note: The expression can also be obtained as shown below:

The alternate a's and b's can be generated using one of the following ways:

1. (ab)*
2. b(ab)*
3. (ba)*
4. a(ba)*

So, the regular expression to generate alternate a's and b's can be obtained by taking the union of all four types of regular expressions as shown below:

(ab)* + b(ab)* + (ba)* + a(ba)*

Example 3.2: Obtain a regular expression to accept a language consisting of strings of 0's and 1's with at most one pair of consecutive 0's

It is clear from the statement that the string consisting of at most one pair of consecutive 0's may

1. begin with combination of any number of 1's and 01's represented by (1+01)*
2. end with any number of 1's represented by 1*

So, the complete regular expression which consists of strings 0's and 1's with at most one pair of consecutive 0's is given by

(1 + 01)* 00 1*

Example 3.3: Obtain a regular expression to accept a language containing at least one a and at least one b where $\Sigma = \{a, b, c\}$

Strings of a's b's and c's can be generated using the regular expression

(a + b + c)*

But this string should have atleast one 'a' and at least one 'b'. There are two cases to be considered:

1. first 'a' preceding 'b' which can be represented using
   c*a(a+c)*b

2. first 'b' preceding 'a' which can be represented using
   c*b(b+c)*a

The regular expression (a + b + c)* can be preceded by one of the regular expressions considered in the two cases just discucced. So, the final regular expression is

c*a(a+c)*b(a+b+c)* + c*b(b+c)*a(a+b+c)*

This expression can also be written as shown below:

[c*a(a+c)*b + c*b(b+c)*a] (a+b+c)*

Example 3.4: Obtain a regular expression to accept a language consisting of strings of a's and b's of even length.

String of a's and b's of even length can be obtained by the combination of the strings aa, ab, ba and bb. The language may even consist of an empty string denoted by ε. So, the regular expression can be of the form

(aa + ab + ba + bb)*

The * closure includes the empty string.

Note: The language corresponding to the regular expression is denoted by

$$L(R) = \{(aa + ab + ba + bb)^n \mid n \geq 0\}$$

Example 3.5: Obtain a regular expression to accept a language consisting of strings of a's and b's of odd length.

String of a's and b's of odd length can be obtained by the combination of the strings aa, ab, ba and bb followed by either *a* or *b*. So, the regular expression can be of the form

(aa + ab + ba + bb)* (a+b)

String of a's and b's of odd length can also be obtained by the combination of the strings aa, ab, ba and bb preceded by either *a* or *b*. So, the regular expression can also be represented as

(a+b) (aa + ab + ba + bb)*

Note: Even though these two expression are seems to be different, the language corresponding to those two expression is same. So, a variety of regular expressions can be obtained for a language and all are equivalent.

Example 3.6: Obtain a regular expression such that L(R) = {w | w ∈ {0, 1}* with at least three consecutive 0's

An arbitrary string consisting of 0's and 1's can be represented by the regular expression

(0+1)*

This arbitrary string can precede three consecutive zeros and can follow three consecutive zeros. So, the regular expression can be written as

$$(0+1)^* 000 (0+1)^*$$

Note: The language corresponding to the regular expression can be written as

$$L(R) = \{ (0+1)^m 000(0+1)^n \mid m \geq 0 \text{ and } n \geq 0\}$$

Example 3.7: Obtain a regular expression to accept strings of a's and b's ending with 'b' and has no substring aa

Note: The statement "strings of a's and b's ending with 'b' and has no substring aa" can be restated as "string made up of either b or ab". Note that if we state something like this, the substring aa will never occur in the string and the string ends with 'b'. So, the regular expression can be of the form

$$(b + ab)^*$$

But, because of * closure, even null string is also included. But, the string should end with 'b'. So, instead of * closure, we can use positive closure '+'. So, the regular expression to accepc strings of a's and b's ending with 'b' and has no substring aa can be written as

$$(b+ab)^+$$

The above regular expression can also be written as

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

Note: The language corresponding to the regular expression can be written as

$$L(R) = \{ (b + ab)^n \mid n \geq 1 \}$$

Example 3.8: Obtain a regular expression to accept strings of 0's and 1's having no two consecutive zeros.

The first observation from the statement is that whenever a 0 occurs it should be followed by 1. But, there is no restriction on the number of 1's. So, it is a string consisting of any combinations of 1's and 01's. So, the partial regular expression for this can be of the form

$$(1 + 01)^*$$

No doubt that the above expression is correct. But, suppose the string ends with a 0. What to do? For this, the string obtained from above regular expression may end with 0 or may end with ε (i.e., may not end with 0). So, the above regular expression can be written as

$$(1 + 01)^* (0 + \varepsilon)$$

Example 3.9: Obtain a regular expression to accept a language consisting of 0's and 1's such that every pair of adjacent 0's appears before any pair of adjacent 1's.

Strings of 0's and 1's having no two consecutive zeroes is given by

$$(1 + 01)^* (0 + \varepsilon)$$

Strings of 0's and 1's having no two consecutive ones is given by

$$(0 + 10)^* (1 + \varepsilon)$$

The string having no two consecutive ones followed by string having no two consecutive zeroes is obtained by concatenating the first with the second regular expression as shown below:

$$(0 + 10)^* (1 + \varepsilon) (1 + 01)^* (0 + \varepsilon)$$

In the above regular expression, it is clear that every pair of adjacent 0's appears before any pair of adjacent 1's.

Example 3.10: Obtain a regular expression to accept strings of a's and b's of length ≤ 10

The regular expression for this can be written as

$$\varepsilon + a + b + aa + ab + ba + bb + \ldots\ldots. + bbbbbbbbba + bbbbbbbbbb$$

But, using .... in a regular expression is not recommended and so we can write the above expression as

$$(\varepsilon + a + b)^{10}$$

Example 3.11: Obtain a regular expression to accept string of a's and b's starting with 'a' and ending with 'b'.

Strings of a's and b's of arbitrary length can be written as

$$(a + b)^*$$

But, the string should start with 'a' and end with 'b'. So, the regular expression can be written as

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

Example 3.12: Obtain a regular expression to accept string of a's and b's whose tenth symbol from the right end is a.

Since we are interested only from the right end we can have a regular expression as shown below:

a (a + b) (a + b) (a + b) (a + b) (a + b) (a + b) (a + b) (a + b) (a + b)

It is clear from the above expression that the string has a strings of a's and b's whose tenth symbol from the right end is *a*. But, the string can be preceded by any number of a's and b's. So, the regular expression can be written as

(a + b)*a (a + b) (a + b) (a + b) (a + b) (a + b) (a + b) (a + b) (a + b) (a + b)

It is clear from this expression that all strings must be of length 10 or more. Here, we need to track the last 10 characters.

Example 3.13: Obtain a regular expression to accept the words with two or more letters but beginning and ending with the same letter where $\Sigma = \{a, b\}$

The string consisting of a's and b's can be denoted by the regular expression

(a + b)*

Since, the string should start and end with the same letter, the above regular expression can be enclosed between 'a' and 'a' or 'b' and 'b'. The regular expression for the same can be denoted by

a (a + b)* a + b(a + b)* b

This regular expression indicates that the string may start and end with either 'a' or 'b'.

Example 3.14: Obtain a regular expression to accept strings of a's and b's whose length is either even or multiples of 3 or both.

The regular expression whose length is even can be obtained using

((a + b) (a + b))*

and the regular expression whose length is multiples of 3 can be obtained using

((a + b) (a + b) (a + b))*

Thus, the regular expression whose length is either even or multiples of 3 or both is given by

((a + b) (a + b))* + ((a + b) (a + b) (a + b))*

Example 3.15: Obtain a regular expression to accept strings of a's and b's such that third symbol from the right is *a* and fourth symbol from the right is *b*

We are interested only in the third and fourth symbol so that third symbol from the right end is *a* and fourth symbol from the right end is b and the regular expression for this is given by

$$ba(a + b)(a + b)$$

But, this substring can be preceded by any combinations of a's and b's which is given by the regular expression

$$(a + b)^*$$

By concatenating the above two regular expressions, we can write the regular expression to accept the given language as

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

Example 3.16: Obtain a regular expression to accept strings of a's and b's such that every block of four consecutive symbols contains at least two a's.

This clearly indicates that always we are interested in block of four consecutive symbols and see that at least two a's are present. Let A, B, C, D are the symbols in a block of four consecutive symbol and minimum two symbols out of A, B, C and D should have two a's as shown below:

AB (Here, A and B are a's where as C and D $\in$ {a, b}) can be represented as
$$aa(a+b)(a+b)$$

AC (Here, A and C are a's where as B and D $\in$ {a, b}) can be represented as
$$a(a+b)a(a+b)$$

AD (Here, A and D are a's where as B and C $\in$ {a, b}) can be represented as
$$a(a+b)(a+b)a$$

BC (Here, B and C are a's where as A and D $\in$ {a, b}) can be represented as
$$(a+b)aa(a+b)$$

BD (Here, B and D are a's where as A and C $\in$ {a, b}) can be represented as
$$(a+b)a(a+b)a$$

CD (Here, C and D are a's where as A and B $\in$ {a, b}) can be represented as
$$(a+b)(a+b)aa$$

Note: Since it is specified that at least two a's are present, it is not possible to have star operator * (which indicates zero or more symbols) since it encompasses even the empty string. So, we have to use the + operator (which indicates one or more symbols) and can be expressed as shown below:

$[aa(a+b)(a+b) + a(a+b)a(a+b) + a(a+b\ (a+b)a + (a+b)aa(a+b) + (a+b)a\ (a+b)a + (a+b)(a+b)aa]^+$

3.2 Relation between FA and Regular Expression

This section shows the relation between DFA, NFA, ε-NFA and regular expression and how to obtain an ε-NFA from the regular expression. Once we have an ε-NFA, we can easily construct a DFA and we know that any language accepted by a DFA is regular. Since a DFA is obtained from an ε-NFA or from NFA, we can say that any language accepted by an ε-NFA or NFA is also regular. Since an ε-NFA is obtained from a regular expression, we can say that a regular language in fact can be denoted by a regular expression. So, given a regular expression we can obtain an ε-NFA, from ε-NFA we have already seen the way to obtain a DFA (section 2.16). In fact, regular expression, ε-NFA, NFA and DFA all represent the same and but notations are different. The diagram showing the different notations and the plan showing the relations is shown in figure 3.1.

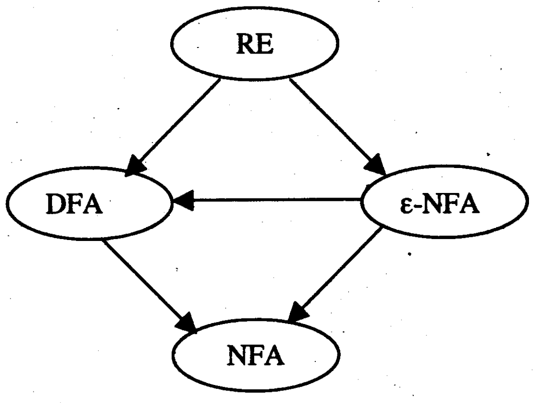


Figure 3.1 Various notations to represent FA and their relations

Let us see how to obtain an ε-NFA from the regular expression and later to obtain the regular expression from a given FA.

3.3 To Obtain ε-NFA from Regular Expression

Theorem: Let R be a regular expression. Then there exists a finite automaton $M=(Q, \Sigma, \delta, q_0, F)$ which accepts L(R).

Proof: By definition, ϕ, ε and *a* are regular expressions. So, the corresponding machines to recognize the language for the respective expressions are shown in figure 3.2.a, 3.2.b and 3.2.c respectively.

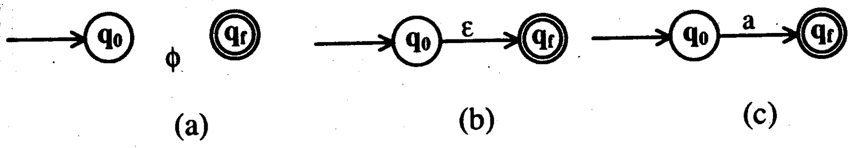


Fig 3.2 ε-NFAs to accept ϕ, ε and a

The schematic representation of a regular expression R to accept the language L(R) is shown in figure 3.3. where q is the start state and f is the final state of machine M.

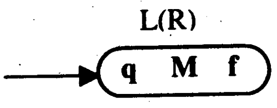


Fig 3.3 Schematic representation of FA accepting L(R)

In the definition of a regular expression it is clear that if R and S are regular expression, then R+S and R.S and R* are regular expressions which clearly uses three operators '+', '*' and '.'. Let us take each case separately and construct equivalent machine.

Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, f_1)$ be a machine which accepts the language $L(R_1)$ corresponding to the regular expression R_1. Let $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, f_2)$ be a machine which accepts the language $L(R_2)$ corresponding to the regular expression R_2. Then the various machines corresponding to the regular expressions $R_1 + R_2$, $R_1 . R_2$ and R* are shown below:

Case 1: $R = R_1 + R_2$. We can construct an NFA which accepts either $L(R_1)$ or $L(R_2)$ which can be represented as $L(R_1 + R_2)$ as shown in figure 3.4.

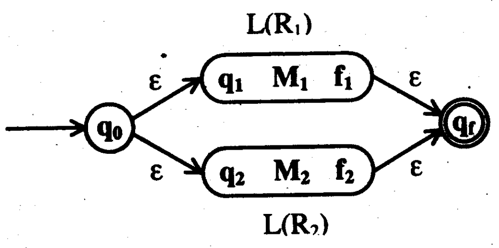


Fig. 3.4 To accept the language L(R1 + R2)

It is clear from figure 3.4 that the machine can either accept $L(R_1)$ or $L(R_2)$. Here, q_0 is the start state of the combined machine and q_f is the final state of combined machine M.
Case 2: $R = R_1 . R_2$. We can construct an NFA which accepts $L(R_1)$ followed by $L(R_2)$ which can be represented as $L(R_1 . R_2)$ as shown in figure 3.5.

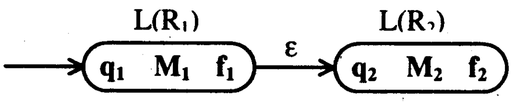


Fig. 3.5To accept the language L(R1 . R2)

It is clear from figure 3.5 that the machine after accepting $L(R_1)$ moves from state q_1 to f_1. Since there is a ε-transition, without any input there will be a transition from state f_1 to state q_2. In state q_2, upon accepting $L(R_2)$, the machine moves to f_2 which is the final state. Thus, q_1 which is the start state of machine M_1 becomes the start state of the combined machine M and f_2 which is the final state of machine M_2, becomes the final state of machine M and accepts the language $L(R_1.R_2)$.

Case 3: $R = (R_1)^*$. We can construct an NFA which accepts either $L(R_1)^*)$ as shown in figure 3.6.

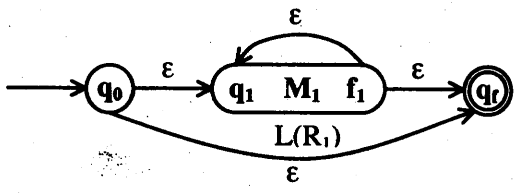


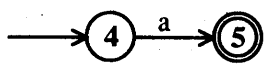
Fig. 3.6 To accept the language L(R1)*

It is clear from figure 3.6 that the machine can either accept ε or any number of $L(R_1)$s thus accepting the language $L(R_1)^*$. Here, q_0 is the start state q_f is the final state.

Example 3.17: Obtain an NFA which accepts strings of a's and b's starting with the string ab.

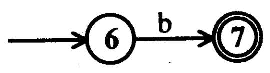
The regular expression corresponding to this language is ab(a+b)*.

Step 1: The machine to accept 'a' is shown below.

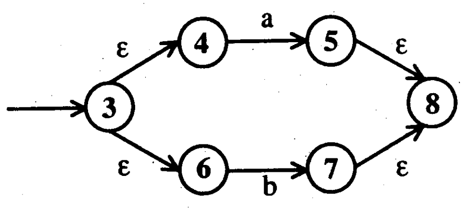


$$\begin{tikzcd} \arrow[r] & 4 \arrow[r, "a"] & \boxed{5} \end{tikzcd}$$

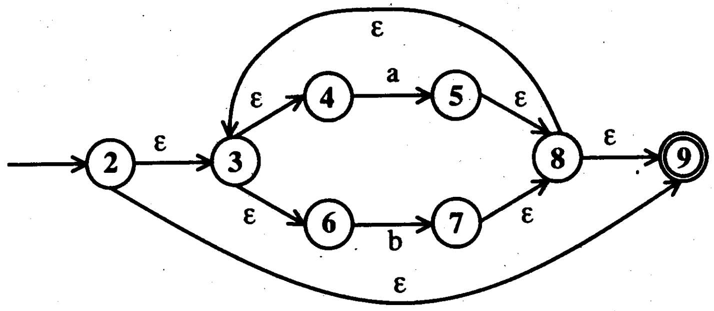
Step 2: The machine to accept 'b' is shown below.



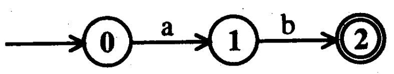
Step 3: The machine to accept (a + b) is shown below.



Step 4: The machine to accept (a+b)* is shown below.



Step 5: The machine to accept ab is shown below.



Step 6: The machine to accept ab(a+b)* is shown below.

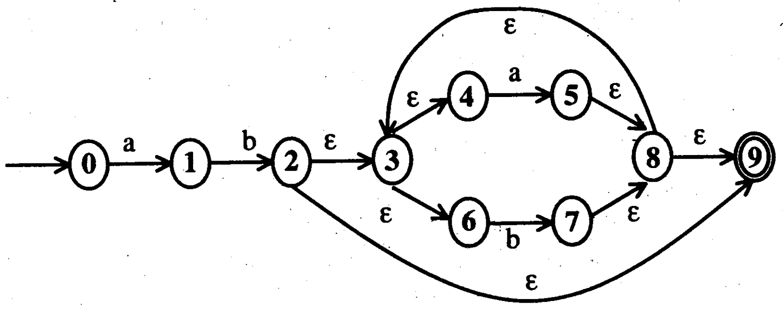
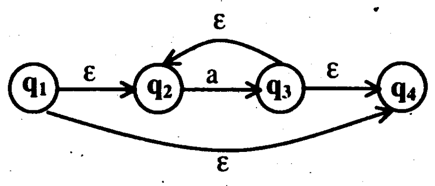


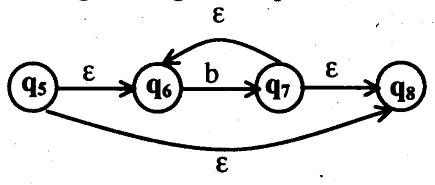
Fig. 3.7 To accept the language L(ab(a+b)*)

Example 3.18: Obtain an NFA for the regular expression a* + b* + c*

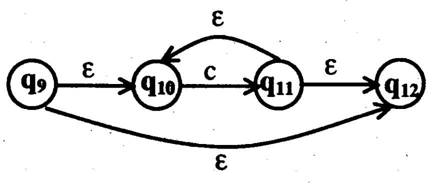
The machine corresponding the regular expression a* can be written as



The machine corresponding the regular expression b* can be written as



The machine corresponding the regular expression c* can be written as



The machine corresponding the regular expression a* + b* + c* is shown in figure 3.8.

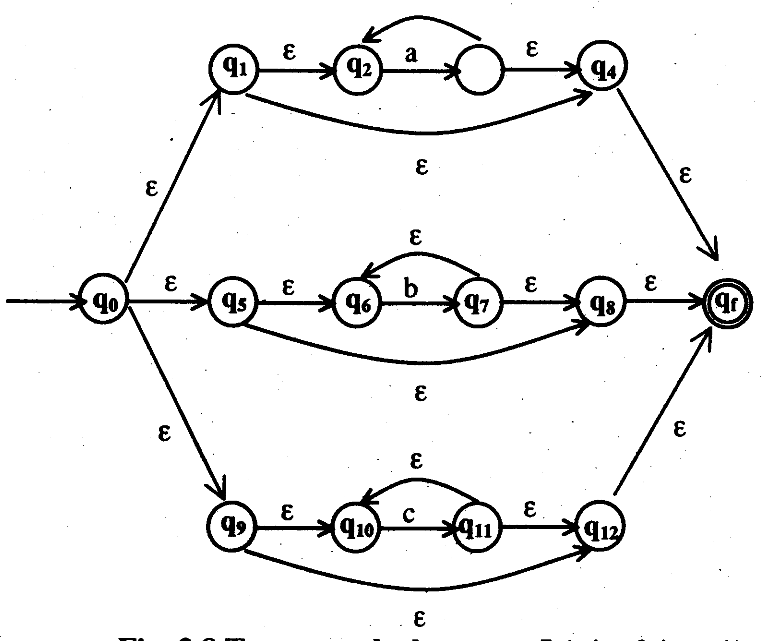
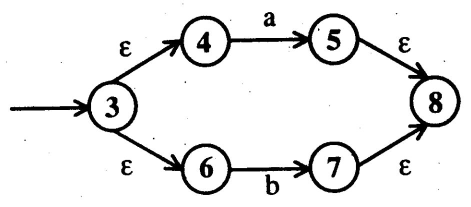


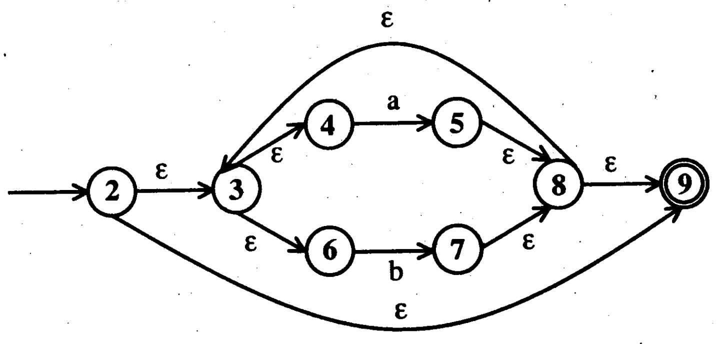
Fig. 3.8 To accept the language L(a* + b* + c*)

Example 3.19: Obtain an NFA for the regular expression (a+b)*aa(a+b)*

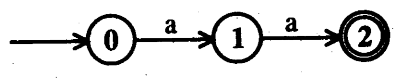
Step 1: The machine to accept (a + b) and (a+b)* are shown below.



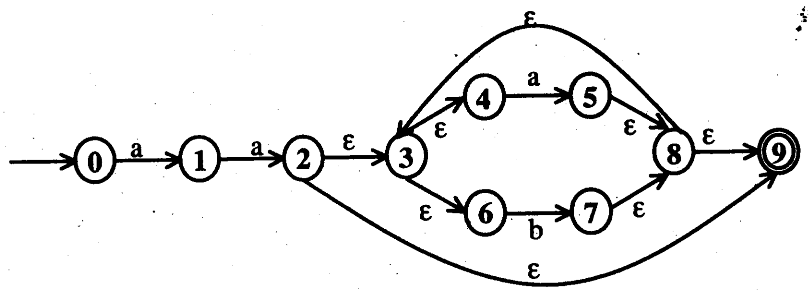
Step 2: The machine to accept (a+b)* is shown below.



Step 3: The machine to accept aa is shown below.



Step 4: The machine to accept aa(a+b)* is shown below.



Step 5: The machine to accept (a+b)*aa(a+b)* is shown in figure 3.9.

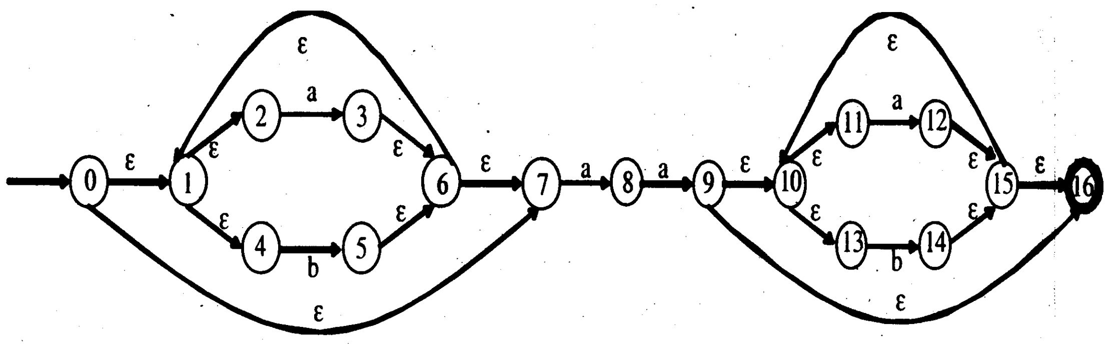


Fig. 3.9 NFA to accept $(a+b)^*aa(a+b)^*$

3.4 To Obtain RE from FA(Kleene's theorem)

In this section let us see how to obtain a regular expression from FA using Kleene's theorem.

Theorem: Let $M = (Q, \Sigma, \delta, q_1, F)$ be an FA recognizing the language L. Then there exists an equivalent regular expression R for the regular language L such that $L = L(R)$.

Note: The proof along with the procedure to obtain a regular expression from finite automaton is shown below:

Proof: Let $Q = \{q_1, q_2, \ldots q_n\}$ are the states of machine M where *n* is the number of states. The path from state to *i* to state *j* through an intermediate state whose number is not greater than *k* is given by the regular expression R_{ij}^k as shown below:

$$R_{ij}^k = \{w \in \Sigma^* \mid \text{w is label (path) from } i \text{ to state } j \text{ that goes through an intermediate state whose number is not greater than } k\}$$

where $i > k$ and $j > k$. The string *w* can be written as

$$w = xy$$

where $|x| > 0$ and $|y| > 0$ and $\hat{\delta}(i, x) = k$ and $\hat{\delta}(k,y) = j$

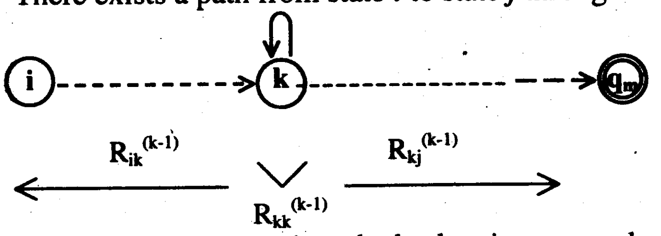
Basis: $k = 0$. This indicates that there is no intermediate state and the path from state *i* to state *j* is given by the following two conditions:

1. There is a direct edge from state *i* to state *j*. This is possible when $i \neq j$. Here, a DFA M with all input symbols *a* such that there is a transition from state *i* to state *j* is considered with following cases:
   a. No input symbol and the corresponding regular expression is given by
   $$R_{ij}^{(0)} = \phi$$
   b. Exactly one input symbol *a* on which there is a transition from state *i* to state to *j* and the corresponding regular expression is given by
   $$R_{ij}^{(0)} = a$$
   c. There are multiple inputs $a_1, a_2, a_3 \ldots a_k$ where there is a transition from each symbol from state *i* to state to *j* and the corresponding regular expression is given by
   $$R_{ij}^{(0)} = a_1 + a_2 + a_3 + \ldots. + a_k$$

2. There is only one state such that i = j and there exists a path from *i* to itself on input symbol *a* forming a self loop or a path of length 0 (i.e., if no loop path from state *i* to state *i* then there is a path of length 0) and is denoted by ε. Thus the regular expressions corresponding to various cases for 1.a, 1.b and 1.c is given by
$$R_{ij}^{(0)} = \phi + \varepsilon$$
$$R_{ij}^{(0)} = a + \varepsilon$$
$$R_{ij}^{(0)} = a_1 + a_2 + a_3 + \ldots. + a_k + \varepsilon$$

Induction: Suppose, there exists a path from *i* to *j* through a state which is not higher than *k*. This situation leads to two cases:

1. There exists a path from state *i* to state *j* which does not go through *k* and so the language accepted is $R_{ij}^{(k-1)}$.
2. There exists a path from state *i* to state *j* through *k* as shown below:



The path from state *i* to state *j* can be broken into several pieces as shown below:
1. The path from state *i* to state *k* and not passing through a state higher than *k* is given by
$$R_{ik}^{(k-1)}$$

2. The path from state k to state k and not passing through a state higher than k (may exist zero or more times) is given by

   $(R_{kk}^{(k-1)})^*$

3. The path from state k to state j and not passing through a state higher than k is given by

   $R_{kj}^{(k-1)}$

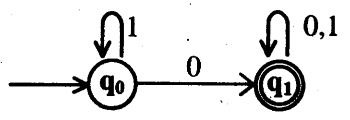
So, the regular expression for the path from state i to state j through no state higher than k is given by the concatenation of above 3 regular expressions as shown below:

$$R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} (R_{kk}^{(k-1)})^* R_{kj}^{(k-1)}$$

By constructing the regular expressions in increasing order of subscripts, each $R_{ij}^{(k)}$ depends only on expressions with a smaller superscript and all the regular expressions are available whenever there is a need. Finally, we will have $R_{ij}^{(n)}$ for all i and j.

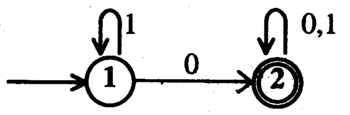
Note: Assume that state 1 is the initial state and the regular expression for the language is the sum of all regular expressions $R_{ij}^{(n)}$ where state j is an accepting state.

Example 3.20: Obtain a regular expression for the FA shown below:



What is the language corresponding to the regular expression?

Solution: Let $q_0 = 1$ and $q_1 = 2$. By renaming the states, the above FA can be written as shown below:



By following the procedure shown in Kleene's theorem (section 3.4) we have

Basis: when $k = 0$

$R_{11}^{(0)} = \varepsilon + 1$
$R_{12}^{(0)} = 0$
$R_{21}^{(0)} = \phi$
$R_{22}^{(0)} = \varepsilon + 0 + 1$

Note: If the beginning and ending states are same add ε which denotes the length 0

Induction: The regular expression corresponding to the path from state *i* to state *j* through a state which is not higher than *k* is given by

$$R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} [R_{kk}^{(k-1)}]^* R_{kj}^{(k-1)}$$

When k = 1 (i.e., path from state *i* to state *j* through a state not higher than 1): The various regular expressions obtained are shown below:

$$\begin{aligned} R_{11}^{(1)} &= R_{11}^{(0)} + R_{11}^{(0)} [R_{11}^{(0)}]^* R_{11}^{(0)} \\ &= (\varepsilon + 1) + (\varepsilon + 1)(\varepsilon + 1)^* (\varepsilon + 1) \\ &= (\varepsilon + 1) + (\varepsilon + 1)\, 1^* (\varepsilon + 1) \\ &= (\varepsilon + 1) + 1^* \\ &= 1^* \end{aligned}$$

$$\begin{aligned} R_{12}^{(1)} &= R_{12}^{(0)} + R_{11}^{(0)} [R_{11}^{(0)}]^* R_{12}^{(0)} \\ &= 0 + (\varepsilon + 1)(\varepsilon + 1)^* \, 0 \\ &= 0 + 1^* 0 \\ &= 1^* 0 \end{aligned}$$

$$\begin{aligned} R_{21}^{(1)} &= R_{21}^{(0)} + R_{21}^{(0)} [R_{11}^{(0)}]^* R_{11}^{(0)} \\ &= \phi + \phi (\varepsilon + 1)^* (\varepsilon + 1) \\ &= \phi \end{aligned}$$

$$\begin{aligned} R_{22}^{(1)} &= R_{22}^{(0)} + R_{21}^{(0)} [R_{11}^{(0)}]^* R_{12}^{(0)} \\ &= (\varepsilon + 0 + 1) + \phi (\varepsilon + 1)^* \, 0 \\ &= (\varepsilon + 0 + 1) \end{aligned}$$

When k = 2 (i.e., path from state *i* to state *j* through a state not higher than 2): The various regular expressions are given by

$$\begin{aligned} R_{11}^{(2)} &= R_{11}^{(1)} + R_{12}^{(1)} [R_{22}^{(1)}]^* R_{21}^{(1)} \\ &= 1^* + 1^* 0 (\varepsilon + 0 + 1)^* \phi \\ &= 1^* \end{aligned}$$

$$\begin{aligned} R_{12}^{(2)} &= R_{12}^{(1)} + R_{12}^{(1)} [R_{22}^{(1)}]^* R_{22}^{(1)} \\ &= 1^* 0 + 1^* 0 (\varepsilon + 0 + 1)^* (\varepsilon + 0 + 1) \\ &= 1^* 0 + 1^* 0 (0 + 1)^* (\varepsilon + 0 + 1) \\ &= 1^* 0 + 1^* 0 (0 + 1)^* \\ &= 1^* 0 (0 + 1)^* \end{aligned}$$

$$\begin{aligned}R_{21}^{(2)} &= R_{21}^{(1)} + R_{22}^{(1)} [R_{22}^{(1)}]^* R_{21}^{(1)}\\ &= \phi + (\varepsilon + 0 + 1)(\varepsilon + 0 + 1)^* \phi\\ &= \phi\end{aligned}$$

$$\begin{aligned}R_{22}^{(2)} &= R_{22}^{(1)} + R_{22}^{(1)} [R_{22}^{(1)}]^* R_{22}^{(1)}\\ &= (\varepsilon + 0 + 1) + (\varepsilon + 0 + 1)(\varepsilon + 0 + 1)^* (\varepsilon + 0 + 1)\\ &= (\varepsilon + 0 + 1) + (0 + 1)^*\\ &= (0 + 1)^*\end{aligned}$$

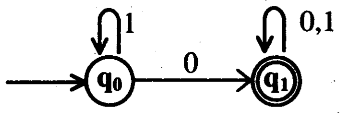
Note: Since the total number of states in the given machine is 2, maximum value of k should be 2.

Since the start state is 1 and final state is 2, the regular expression is given by

$$R_{12}^{2} = 1^*0\,(0 + 1)^*$$

So, the regular expression for the given DFA is $1^*0\ (0 + 1)^*$ which is the language consisting of any number of 1's followed by a zero and then followed by strings of 0's and 1's. This can also be expressed as strings of 0's and 1's with at least one zero.

Example 3.21: Obtain a regular expression for the FA shown below:



What is the language corresponding to the regular expression?

Note: The solution for this problem is already given in previous example. Another approach to solve this problem is that, instead of calculating the regular expressions for R_{ij} **from** $k = 0$ to n, start from $k = n$, and then work back to the case when $k = 0$.

In the current example, number of states $n = 2$ and hence to start with assume $k = 2$. The regular expression is given by

$$R_{12}^{(2)} = R_{12}^{(1)} + R_{12}^{(1)} [R_{22}^{(1)}]^* R_{22}^{(1)} \ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(Eq. 1)}$$

It is clear from the above expression that $R_{12}^{(1)}$ and $R_{22}^{(1)}$ are required and are obtained using the following regular expressions:

$$\begin{aligned}R_{12}^{(1)} &= R_{12}^{(0)} + R_{11}^{(0)} [R_{11}^{(0)}]^* R_{12}^{(0)} \ldots\ldots\ldots\ldots\ldots\ldots \text{(Eq. 2)}\\ R_{22}^{(1)} &= R_{22}^{(0)} + R_{21}^{(0)} [R_{11}^{(0)}]^* R_{12}^{(0)} \ldots\ldots\ldots\ldots\ldots\ldots \text{(Eq. 3)}\end{aligned}$$

The regular expressions for these two equations can be obtained if we know $R_{11}^{(0)}$, $R_{12}^{(0)}$, $R_{21}^{(0)}$ and $R_{22}^{(0)}$ which are obtained when k = 0 as shown below:

When k = 0:

$R_{11}^{(0)} = \varepsilon + 1$
$R_{12}^{(0)} = 0$
$R_{21}^{(0)} = \phi$
$R_{22}^{(0)} = \varepsilon + 0 + 1$

Note: If the beginning and ending states are same add ε which denotes the length 0

Substituting these values in Eq. 2 and Eq. 3 we have,

$$\begin{aligned} R_{12}^{(1)} &= R_{12}^{(0)} + R_{11}^{(0)}\ [R_{11}^{(0)}]^* R_{12}^{(0)} \\ &= 0 + (\varepsilon + 1)\,(\varepsilon + 1)^*\ 0 \\ &= 0 + 1^*\, 0 \\ &= 1^*0 \end{aligned}$$

$$\begin{aligned} R_{22}^{(1)} &= R_{22}^{(0)} + R_{21}^{(0)}\,[R_{11}^{(0)}]^* R_{12}^{(0)} \\ &= (\varepsilon + 0 + 1) + \phi\,(\varepsilon + 1)^*\ 0 \\ &= (\varepsilon + 0 + 1) \end{aligned}$$

Substituting for $R_{12}^{(1)}$ and $R_{22}^{(1)}$ in Eq. (1) we have

$$\begin{aligned} R_{12}^{(2)} &= R_{12}^{(1)} + R_{12}^{(1)}\,[R_{22}^{(1)}]^* R_{22}^{(1)} \\ &= 1^*0 + 1^*0\,(\varepsilon + 0 + 1)^*\,(\varepsilon + 0 + 1) \\ &= 1^*0 + 1^*0\,(0 + 1)^*\,(\varepsilon + 0 + 1) \\ &= 1^*0 + 1^*0\,(0 + 1)^* \\ &= 1^*0\,(0 + 1)^* \end{aligned}$$

So, the regular expression for the given DFA is $1^*0\ (0 + 1)^*$ which is the language consisting of any number of 1's followed by a zero and then followed by strings of 0's and 1's. This can also be expressed as strings of 0's and 1's with at least one zero.

Example 3.22: Consider the DFA shown below

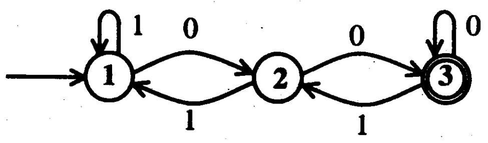
	Σ	
States	0	1
$\rightarrow q_1$	q_2	q_1
q_2	q_3	q_1
$*q_3$	q_3	q_2

Obtain the regular expressions $R_{ij}^{(0)}$, $R_{ij}^{(1)}$ and simplify the regular expressions as much as possible.
Note: The symbol * preceding state q_3 indicates that q_3 is the final state.

Solution: Rename the states 1, 2 and 3 in order and the resulting DFA is shown below:

	Σ	
States	0	1
→1	2	1
2	3	1
*3	3	2

where state 3 is the final state and state 1 is the start state. The corresponding transition diagram is shown below:



By following the procedure shown in Kleene's theorem (section 3.4) we have

Basis: when k = 0

$R_{11}^{(0)} = \varepsilon + 1$
$R_{12}^{(0)} = 0$
$R_{13}^{(0)} = \phi$
$R_{21}^{(0)} = 1$
$R_{22}^{(0)} = \phi + \varepsilon = \varepsilon$
$R_{23}^{(0)} = 0$
$R_{31}^{(0)} = \phi$
$R_{32}^{(0)} = 1$
$R_{33}^{(0)} = \varepsilon + 0$

Note: If the beginning and ending states are same add ε which denotes the length 0 (i.e., whenever i = j)

Induction: The regular expression corresponding to the path from state *i* to state *j* through a state which is not higher than *k* is given by

$$R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} [R_{kk}^{(k-1)}]^* R_{kj}^{(k-1)}$$

When k = 1 (i.e., path from state *i* to state *j* through a state not higher than 1): The various regular expressions obtained are shown below:

$$\begin{aligned}
R_{11}^{(1)} &= R_{11}^{(0)} + R_{11}^{(0)} [R_{11}^{(0)}]^* R_{11}^{(0)} \\
&= (\varepsilon + 1) + (\varepsilon + 1) (\varepsilon + 1)^* (\varepsilon + 1) \\
&= (\varepsilon + 1) + (\varepsilon + 1)\, 1^* (\varepsilon + 1) \\
&= (\varepsilon + 1) + 1^* \\
&= 1^*
\end{aligned}$$

$$\begin{aligned}
R_{12}^{(1)} &= R_{12}^{(0)} + R_{11}^{(0)} [R_{11}^{(0)}]^* R_{12}^{(0)} \\
&= 0 + (\varepsilon + 1) (\varepsilon + 1)^* \, 0 \\
&= 0 + 1^* 0 \\
&= 1^* 0
\end{aligned}$$

$$\begin{aligned}
R_{13}^{(1)} &= R_{13}^{(0)} + R_{11}^{(0)} [R_{11}^{(0)}]^* R_{13}^{(0)} \\
&= \phi + (\varepsilon + 1)^* (\varepsilon + 1)\, \phi \\
&= \phi
\end{aligned}$$

$$\begin{aligned}
R_{21}^{(1)} &= R_{21}^{(0)} + R_{21}^{(0)} [R_{11}^{(0)}]^* R_{11}^{(0)} \\
&= 1 + 1\, (\varepsilon + 1)^* (\varepsilon + 1) \\
&= 1 + 11^* = 11^*
\end{aligned}$$

$$\begin{aligned}
R_{22}^{(1)} &= R_{22}^{(0)} + R_{21}^{(0)} [R_{11}^{(0)}]^* R_{12}^{(0)} \\
&= \varepsilon + 1\, (\varepsilon + 1)^* \, 0 \\
&= \varepsilon + 11^* 0
\end{aligned}$$

$$\begin{aligned}
R_{23}^{(1)} &= R_{23}^{(0)} + R_{21}^{(0)} [R_{11}^{(0)}]^* R_{13}^{(0)} \\
&= 0 + 1\, (\varepsilon + 1)^* \, \phi \\
&= 0
\end{aligned}$$

$$\begin{aligned}
R_{31}^{(1)} &= R_{31}^{(0)} + R_{31}^{(0)} [R_{11}^{(0)}]^* R_{11}^{(0)} \\
&= \phi + \phi\, (\varepsilon + 1)^* (\varepsilon + 1) \\
&= \phi
\end{aligned}$$

$$\begin{aligned}
R_{32}^{(1)} &= R_{32}^{(0)} + R_{31}^{(0)} [R_{11}^{(0)}]^* R_{12}^{(0)} \\
&= 1 + \phi\, (\varepsilon + 1)^* \, 0 \\
&= 1
\end{aligned}$$

$$\begin{aligned}
R_{33}^{(1)} &= R_{33}^{(0)} + R_{31}^{(0)} [R_{11}^{(0)}]^* R_{13}^{(0)} \\
&= (\varepsilon + 0) + \phi\, (\varepsilon + 0)^* \, \phi \\
&= (\varepsilon + 0)
\end{aligned}$$

When k = 2 (i.e., path from state *i* to state *j* through a state not higher than 2): The various regular expressions are given by

$R_{11}^{(2)} = R_{11}^{(1)} + R_{12}^{(1)} [R_{22}^{(1)}]^* R_{21}^{(1)}$
$= 1^* + 1^*0\,(\varepsilon + 11^*0)^*\,11^*$
$= 1^* + 1^*0(11^*0)^*11^*$

$R_{12}^{(2)} = R_{12}^{(1)} + R_{12}^{(1)} [R_{22}^{(1)}]^* R_{22}^{(1)}$
$= 1^*0 + 1^*0\,(\varepsilon + 11^*0)^*\,(\varepsilon + 11^*0)$
$= 1^*0 + 1^*0\,(11^*0)^*\,(\varepsilon + 11^*0)$

$R_{13}^{(2)} = R_{13}^{(1)} + R_{12}^{(1)} [R_{22}^{(1)}]^* R_{23}^{(1)}$
$= \phi + 1^*0\,(\varepsilon + 11^*0)^*\,0$
$= 1^*0(11^*0)^*0$

$R_{21}^{(2)} = R_{21}^{(1)} + R_{22}^{(1)} [R_{22}^{(1)}]^* R_{21}^{(1)}$
$= 11^* + (\varepsilon + 11^*0)\,(\varepsilon + 11^*0)^*\,11^*$
$= 11^* + (\varepsilon + 11^*0)\,(11^*0)11^*$

$R_{22}^{(2)} = R_{22}^{(1)} + R_{22}^{(1)} [R_{22}^{(1)}]^* R_{22}^{(1)}$
$= (\varepsilon + 11^*0) + (\varepsilon + 11^*0)\,(\varepsilon + 11^*0)^*\,(\varepsilon + 11^*0)$
$= (\varepsilon + 11^*0) + (\varepsilon + 11^*0)\,(11^*0)^*\,(\varepsilon + 11^*0)$

$R_{23}^{(2)} = R_{23}^{(1)} + R_{22}^{(1)} [R_{22}^{(1)}]^* R_{23}^{(1)}$
$= 0 + (\varepsilon + 11^*0)\,(\varepsilon + 11^*0)^*\,0$
$= 0 + (\varepsilon + 11^*0)\,(11^*0)^*\,0$

$R_{31}^{(2)} = R_{31}^{(1)} + R_{32}^{(1)} [R_{22}^{(1)}]^* R_{21}^{(1)}$
$= \phi + 1(\varepsilon + 11^*0)^*\,11^*$
$= 1(11^*0)^*11^*$

$R_{32}^{(2)} = R_{32}^{(1)} + R_{32}^{(1)} [R_{22}^{(1)}]^* R_{22}^{(1)}$
$= 1 + 1(\varepsilon + 11^*0)^*\,(\varepsilon + 11^*0)$
$= 1 + 1(11^*0)^*\,(\varepsilon + 11^*0)$

$R_{33}^{(2)} = R_{33}^{(1)} + R_{32}^{(1)} [R_{22}^{(1)}]^* R_{23}^{(1)}$
$= (0 + \varepsilon) + 1(11^*0)^*\,0$

The regular expression is given by R_{13} can be calculated as shown below:
$R_{13}^{(3)} = R_{13}^{(2)} + R_{13}^{(2)} [R_{33}^{(2)}]^* R_{33}^{(2)}$

$= 1^*0(11^*0)^*0 + 1^*0(11^*0)^*0\,[(0 + \varepsilon) + 1(11^*0)^*\,0]^*\,(0 + \varepsilon) + 1(11^*0)^*\,0$

3.5 To Obtain RE from FA(by eliminating states)

In this section let us see how to obtain a regular expression from FA.

Theorem: Let M = (Q, Σ, δ, q_0, F) be an FA recognizing the language L. Then there exists an equivalent regular expression R for the regular language L such that L = L(R).

The general procedure to obtain a regular expression from FA is shown below. Consider the generalized graph

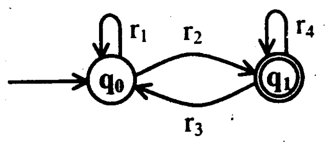


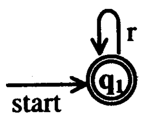
Fig. 3.10 Generalized transition graph

where r_1, r_2, r_3 and r_4 are the regular expressions and correspond to the labels for the edges. The regular expression for this can take the form:

$$r = r_1^* r_2 (r_4 + r_3 r_1^* r_2)^* \qquad (3.1)$$

Note:

1. For each final state (accepting state) q, apply the reduction procedure and bring the graph to the form shown in figure 3.10.
2. If q is not the start state, the reduced graph obtained will be of the form shown in figure 3.10. and use the equation 3.1 to obtain the regular expression.
3. If the start state is also an accepting state, the state elimination process has to be performed so that we should get rid of every state except the start state. The final automaton will be of the form



4. The final regular expression is the sum of all the regular expressions obtained from the reduced automata for each accepting state.

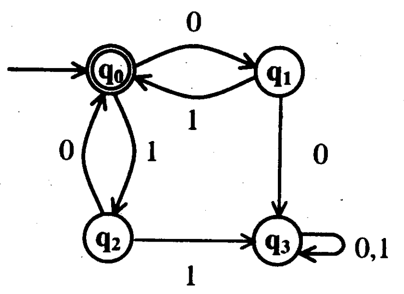
For example, If r_3 is not there in figure 3.10, the regular expression can be of the form

$$r = r_1^* r_2 r_4^* \qquad (3.2)$$

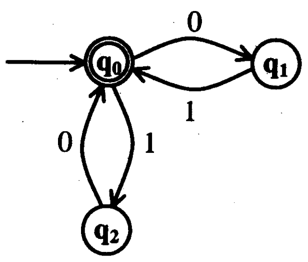
If q_0 and q_1 are the final states, then the regular expression can be of the form

$$r = r_1^* + r_1^* r_2 r_4^* \qquad (3.3)$$

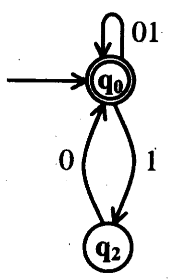
Example 3.23: Obtain a regular expression for the FA shown below:



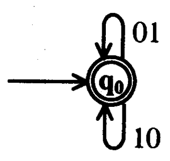
It is clear from the FA that q_3 is the dead state (i.e., once the state q_3 is reached irrespective of the input, the machine stays in q_3 only and there is no way to reach the final state) and so all the edges connected to q_3 can be removed and the resulting figure is shown below:



It is clear from the figure that if we input the string 01, the machine goes to state q_1 and comes back to q_0 and the process can be repeated. So, instead of q_1, we can loop back on the string 01 as shown below:



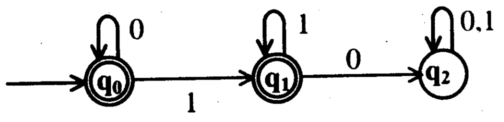
Similarly on 10, the machine comes back to q_0 and so we can replace it by another loop with the edge labeled 10 as shown:



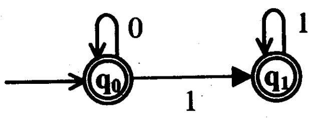
It is clear from this figure that the machine accepts strings of 01's and 10's of any length and the regular expression can be of the form

$$(01 + 10)^*$$

Example 3.24: What is the language accepted by the following FA



Since, state q_2 is the dead state, it can be removed and the following FA is obtained.



The state q_0 is the final state and at this point it can accept any number of 0's which can be represented using notation as

$$0^*$$

q_1 is also the final state. So, to reach q_1 one can input any number of 0's followed by 1 and followed by any number of 1's and can be represented as

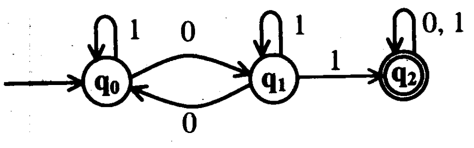
$$0^*11^*$$

So, the final regular expression is obtained by adding 0^* and 0^*11^*. So, the regular expression is

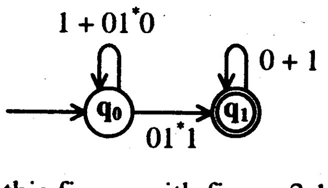
$$\begin{aligned} R.E &= 0^* + 0^*11^* \\ &= 0^*(\varepsilon + 11^*) \\ &= 0^*(\varepsilon + 1^+) \\ &= 0^*(1^*) = 0^*1^* \end{aligned}$$

It is clear from the regular expression that language consists of any number of 0's (possibly ε) followed by any number of 1's(possibly ε).

Example 3.25: Obtain a regular expression for the FA shown below:



The graph should be converted into generalized graph (shown in figure 3.10) by eliminating state q_1 as shown below.



By comparing this figure with figure 3.10, we have

$r_1 = 1 + 01^*0$
$r_2 = 01^*1$
$r_3 = \phi$
$r_4 = 0 + 1$

By substituting these in equation 3.1 we have

$$\begin{aligned} r &= (1 + 01^*0)^* 01^*1[(0+1) + \phi(1 + 01^*0)^*01^*1]^* \\ &= (1 + 01^*0)^* 01^*1[(0+1) + \phi]^* \\ &= (1 + 01^*0)^* 01^*1(0+1)^* \end{aligned}$$

So, the regular expression for the given FA is $(1 + 01^*0)^* 01^*1 (0+1)^*$

3.6 Applications of Regular Expressions

In section 1.20 we have seen how an identifier for a programming language can be represented using the grammar. In section 1.21 we have seen how a finite automaton can be constructed to identify an identifier. In this section, let us see how a regular expression can be formed for an identifying an identifier. This shows that regular expressions, regular grammars and finite automata are equivalent i.e., given one we can find the other. The various applications of regular expressions are provided below:

Regular expressions in UNIX: Regular expressions are extensively used in UNIX operating system. But, certain short hand notations are used in UNIX platform using which complex regular expressions are avoided. For example, the symbol '.' stands for any character, the sequence [abcde.....] stands for the regular expression "a + b + c + d + e.......", the operator | is used in place of +, the operator ? means "zero or one of" etc. Most of the commands are invoked invariably uses regular expressions. For example, grep (Global search for **R**egular Expression and **P**rint) used to search for a pattern of string.

Pattern Matching refers to a set of objects with some common properties. We can match an identifier or a decimal number or we can search for a string in the text.

Lexical analysis Regular expressions are extensively used in the design of lexical analyzer phase. This phase scans the source program and recognizes all the tokens which are logically together. The UNIX commands such as *lex* accepts regular expressions as the input and produces the lexical analyzer generator. This generator takes a high-level description of a lexical analyzer as the input and produces lexical analyzer.

Example 3.26: Obtain a regular expression to identify an identifier.

An identifier stats with a letter. This letter can be followed by combination of zero or more letters and digits i.e., an identifier can be a single letter followed by strings of letters and digits of any length and can be represented as

$$\text{letter (letter + digit)}^*$$

Example 3.27: Obtain a regular expression to identify an integer

An integer can start with any of the signs +, - or ε (null string means no sign) followed by one or more digits ranging from 0 to 9. This can be represented using a regular expression as

$$sd^+ \text{ or } sdd^*$$

where *s* stands for sign and *d* stands for the digits from 0 to 9.

Example 3.28: An application of regular expression in UNIX editor ed.

In UNIX operating system, we can use the editor *ed* to search for a specific pattern in the text. For example, if the command specified is

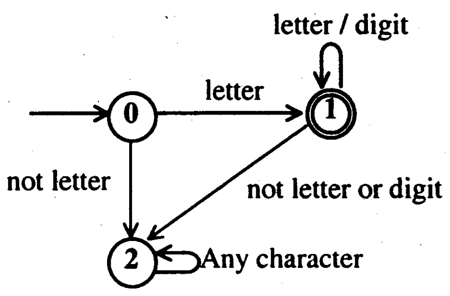
```
/acb*c/
```

then the editor searches for a string which starts with *ac* followed by zero or more b's and followed by the symbol *c*. Note that the editor ed accepts the regular expression and searches for that particular pattern in the text. As the input can vary dynamically, it is challenging to write programs for string patters of these kinds.

Note: To write an algorithm for pattern matching what we have to do is, first obtain the regular expression for the pattern and obtain an NFA as discussed in section 3.3. The NFA can be converted into its equivalent DFA as discussed in section 2.10. The resulting DFA can be reduced as discussed in section 2.11 and obtain the transitional table. By looking at the transitional table or DFA we can easily write an algorithm or the program.

Example 3.29: Write a c program to identifier an identifier.

The FA to accept an identifier is shown in figure below:



To start with the machine will be in state 0. So, initialize the variable state to 0 as shown below:

```
        state = 0;
```

In state 0, if the input symbol is a letter it goes to state 1 else it goes to state 2. In state 1 as long as the input is letter/digit the machine stays in state 1 else it enters into state 2. Once the machine enters into state 2 the string is not a valid identifier. The complete program to check whether the input string is an identifier or not is shown below:

```
char    str[30];        /* Can be valid identifier or invalid identifier*/

int     i = 0;          /* index variable for string */
int     valid = 0;      /* Empty string is not a valid identifier */
int     state = 0;      /* initial state is 0 */
int     over = 1;       /* Enter into while loop initially */

while (over)
{
        switch(state)
        {
                case 0:
                        ch = getch();

                        if (isalpha(ch))
                        {
                                str[i++] = ch;
                                valid = 1;      /* valid identifier */
                                state = 1;      /* So, go to state 1 */
                        }
                        else
                {
                        str[i++] = ch;
                        valid = 0;      /* invalid identifier */
                        state = 2;
                }
                break;
        case 1:
                ch = getch();
                if (isalnum(ch))        /* collect all valid characters
                {
                        str[i++] = ch;  /* store the character */
                        valid = 1;      /* Valid character */
                        state = 1;      /* Remain in state 1 */
                }
                else if (ch == '\r')
                {
                        str[i] = '\0';  /* Null terminate */
                        over = 0;       /* Quit the loop */
                }
                else
                {       /* collect all invalid characters */
                        str[i++] = ch;  /* store the character */
                        valid = 0;      /* invalid identifier */
                        state = 2;      /* goto invalid state */
                }
                break;
        case 2:
                while ( (ch = getch()) != '\r') /*Carriage return*/
                {
                        str[i++] = ch;
                }
                str[i] = '\0';  /* NULL terminate */
                valid = 0;      /* invalid identifier */
                over = 0;       /* Quit the loop */
    }
}

if (valid)
        printf("The string %s is a valid identifier\n", str);
else
        printf("The string %s is not a valid identifier\n", str);
```

3.7 Chomsky Hierarchy

In section 1.14, we have studied the grammar of only one type. There are other types of grammars also. Noam Chomsky who is the founder of formal language theory has classified the grammar into four types

- type 0 grammar
- type 1 grammar
- type 2 grammar
- type 3 grammar

3.8 Type 0 grammar(Unrestricted grammar)

The type 0 grammar or unrestricted grammar is defined as follows.

Definition: A grammar G = (V, T, P, S) is said to be **type 0 grammar** or **unrestricted grammar** if all the productions are of the form $\alpha \rightarrow \beta$ where

$$\alpha \in (V \cup T)^+ \text{ and } \beta \in (V \cup T)^*$$

In this type there are no restrictions on length of α and β. The only restriction is that α can not be ε i.e., ε can not appear on the left hand side of any production. But, ε can appear on the right hand side of the function. This is the largest family of grammars and much more powerful than other types of grammars. Any language can be obtained from this grammar.

The language generated from this grammar is called type 0 language or recursively enumerable language. Only Turing machine can recognize this language. In the subsequent chapters, we shall see how to obtain the Turing machines. Consider the example shown below:

$$\begin{aligned} S &\rightarrow aAb \mid \varepsilon \\ aA &\rightarrow bAA \\ bA &\rightarrow a \end{aligned}$$

is an unrestricted grammar also called *phrase structured grammar.*

3.9 Type 1 grammar or context sensitive grammar

The type 1 grammar or context sensitive grammar is defined as follows.

Definition: A grammar G = (V, T, P, S) is said to be **type1 grammar** or **context sensitive grammar** if all the productions are of the form $\alpha \rightarrow \beta$ as in type 0 grammar. But, there is a restriction on length of β. The length of β must be at least as much as the length of α i.e., $|\beta| \geq |\alpha|$ and α and $\beta \in (V \cup T)^+$ i.e., ε can not appear on the left hand or right hand side of any production. It is an ε-free grammar.

The language generated from this grammar is called type 1 language or context sensitive language. Linear bounded automata (LBA) can be constructed to recognize the language generated from this grammar and we see the construction of LBA in later chapters. The following example shows type 1 grammar or context sensitive grammar.

$$\begin{aligned} S &\rightarrow aAb \\ aA &\rightarrow bAA \\ bA &\rightarrow aa \end{aligned}$$

3.10 Type 2 grammar or context free grammar

The type 2 grammar or context free grammar is defined as follows.

Definition: A grammar G = (V, T, P, S) is said to be type 2 grammar or context free grammar if all the productions are of the form $A \rightarrow \alpha$ where $\alpha \in (V \cup T)^*$ and A is non-terminal. The symbol ε can appear on the right hand side of any production. The context free grammar was discussed in section 1.14. The language generated from this grammar is called type 2 language or context free language. Pushdown automaton (PDA) can be constructed to recognize the language generated from this grammar. The grammar

$$\begin{aligned} S &\rightarrow aB \mid bA \mid \varepsilon \\ A &\rightarrow aA \mid b \\ B &\rightarrow bB \mid a \mid \varepsilon \end{aligned}$$

is an example for context free grammar.

3.11 Type 3 grammar or Regular grammar

The type 3 grammar or context free grammar is defined as follows.

Definition: The grammar G = (V, T, P, S) is said to be **type 3 grammar** or **regular grammar** iff the grammar is right linear or left linear. A grammar G is said to be **right linear** if all the productions are of the form

$$A \rightarrow wB \text{ and/or } A \rightarrow w$$

where $A, B \in V$ and $w \in T^*$. A grammar G is said to be **left linear** if all the productions are of the form

$$A \rightarrow Bw \text{ and/or } A \rightarrow w$$

where $A, B \in V$ and $w \in T^*$. The following grammar

$$\begin{aligned} S &\rightarrow aaB \mid bbA \mid \varepsilon \\ A &\rightarrow aA \mid b \\ B &\rightarrow bB \mid a \mid \varepsilon \end{aligned}$$

is a right linear grammar. Note that ε and string of terminals can appear on RHS of any production and if non-terminal is present on R.H.S of any production, only one non-terminal should be present and it has to be the right most symbol on R.H.S. The grammar

$$\begin{aligned} S &\rightarrow Baa \mid Abb \mid \varepsilon \\ A &\rightarrow Aa \mid b \\ B &\rightarrow Bb \mid a \mid \varepsilon \end{aligned}$$

is a left linear grammar. Note that ε and string of terminals can appear on RHS of any production and if non-terminal is present on R.H.S of any production, only one non-terminal should be present and it has to be the left most symbol on R.H.S.

Consider the grammar

$$\begin{aligned} S &\rightarrow aA \\ A &\rightarrow aB \mid b \\ B &\rightarrow Ab \mid a \end{aligned}$$

In this grammar, each production is either left linear or right linear. But, the grammar is not either left linear or right linear. Such type of grammar is called **linear grammar**. So, a grammar which has at most one non-terminal on the right side of any production without restriction on the position of this non-terminal (note the non-terminal can be leftmost or right most) is called linear grammar.

Note that the language generated from the regular grammar is called regular language. So, there should be some relation between the regular grammar and the FA, since, the language accepted by FA is also regular language. So, we can construct a finite automaton given a regular grammar. Let us discuss how to obtain finite automaton from the regular expression.

3.12 Finite Automaton from Regular Grammar

Theorem: Let G = (V, T, P, S) be a right linear grammar. Then there exists a language L(G) which is accepted by a FA i.e., the language generated from the regular grammar is regular language.

Proof: Let V = $\{q_0, q_1, \ldots\}$ be the variables and the start state S = q_0. Let the productions in the grammar be

$$\begin{aligned}
q_0 &\rightarrow x_1q_1\\
q_1 &\rightarrow x_2q_2\\
q_2 &\rightarrow x_3q_3\\
&\ldots\ldots\\
&\ldots\ldots\\
q_n &\rightarrow x_{n+1}
\end{aligned}$$

Assume that the language L(G) generated from these productions is *w*. Corresponding to each production in the grammar we can have a equivalent transitions in the FA to accept the string *w*. After accepting the string *w*, the FA will be in the final state. The procedure to obtain FA from these productions is given below:

step1:
q_0 which is the start symbol in the grammar is the start state of FA.

step2:
For each production of the form

$$q_i \rightarrow wq_j$$

the corresponding transition defined will be of the form

$$\delta^*(q_i, w) = q_j$$

step3:
For each production of the form

$$q_i \rightarrow w$$

the corresponding transition defined will be of the form

$$\delta^*(q_i, w) = q_f$$

where q_f is the final state.

As the string w ∈ L(G) is also accepted by FA, by applying the transitions obtained step 1 through step3, the language is regular. So, the theorem is proved.

Example 3.30: Construct a DFA to accept the language generated by the following grammar.

S	→	01A
A	→	10B
B	→	0A|11

Note that for each production of the form

$$A \rightarrow wB$$

the corresponding transition will be

$$\delta(A, w) = B$$

Also, for each production of the form

$$A \rightarrow w$$

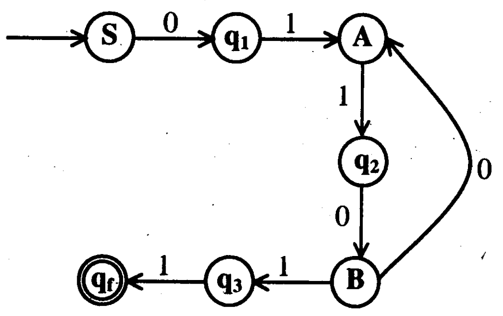
we can introduce the transition

$$\delta(A, w) = q_f$$

where q_f is the final state. The transitions obtained from grammar G is shown using the following table:

Productions	Transitions
$S \rightarrow 01A$	$\delta(S, 01) = A$
$A \rightarrow 10B$	$\delta(A, 10) = B$
$B \rightarrow 0A$	$\delta(B, 0) = A$
$B \rightarrow 11$	$\delta(B, 11) = q_f$

The FA corresponding to the transitions obtained is shown below:



So, the DFA $M = (Q, \Sigma, \delta, q_0, F)$ where

$Q = \{ S, A, B, q_f, q_1, q_2, q_3\}$
$\Sigma = \{0, 1\}$.
$q_0 = S$
$F = \{q_f\}$
δ is as obtained from the table above.

The additional vertices introduced are q_1, q_2, q_3.

Example 3.31: Construct a DFA to accept the language generated by the following grammar.

S	$\rightarrow$	aA | ε
A	$\rightarrow$	aA | bB | ε
B	$\rightarrow$	bB | ε

Note that for each production of the form

$$A \rightarrow wB$$

the corresponding transition will be

$$\delta(A, w) = B$$

Also, for each production of the form

$$A \rightarrow w$$

we can introduce the transition

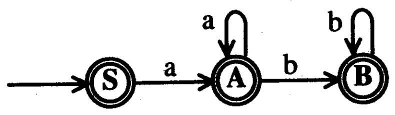
$$\delta(A, w) = q_f$$

where q_f is the final state. The transitions obtained from grammar G is shown using the following table:

Productions	Transitions
S $\rightarrow$ aA	$\delta(S, a) = A$
S $\rightarrow$ ε	S is the final state
A $\rightarrow$ aA	$\delta(A, a) = A$
A $\rightarrow$ bB	$\delta(A, b) = B$
A $\rightarrow$ ε	A is the final state
B $\rightarrow$ bB	$\delta(B, b) = B$
B $\rightarrow$ ε	B is the final state.

Note: For each production of the form A $\rightarrow$ ε, make A as the final state.

The FA corresponding to the transitions obtained is shown below:



So, the DFA M = (Q, Σ, δ, q_0, F) where

Q = {S, A, B}
Σ = {a, b}
q_0 = S

$F = \{S, A, B\}$
δ is as obtained from the table above.

3.13 Regular Grammar from Finite Automaton

Theorem: Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton. If L is the regular language accepted by FA, then there exists a right linear grammar $G = (V, T, P, S)$ so that $L = L(G)$.

Proof: Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automata accepting L where
$Q = \{q_0, q_1, \ldots q_n\}$
$\Sigma = \{a_1, a_2, \ldots a_m\}$

A regular grammar $G = (V, T, P, S)$ can be constructed where
$V = \{q_0, q_1, \ldots\ldots q_n\}$
$T = \Sigma$
$S = q_0$

The productions P from the transitions can be obtained as shown below:

step1:

For each transition of the form $\delta(q_i, a) = q_j$ the corresponding production defined will be

$$q_i \rightarrow aq_j$$

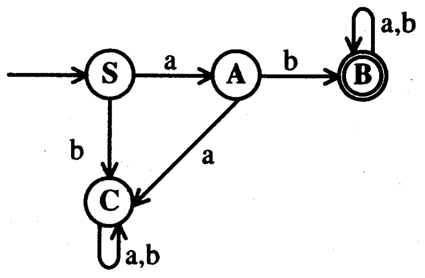
step2:

If $q \in F$ i.e., if q is the final state in FA, then introduce the production

$$q \rightarrow \varepsilon$$

As these productions are obtained from the transitions defined for FA, the language accepted by FA is also accepted by the grammar.

Example 3.32: Construct a regular grammar from the following FA



Note that for each transition of the form δ(A, a) = B, introduce the production A → aB. If q ∈ F i.e., if q is the final state, introduce the production q → ε. The productions obtained from the transitions defined for FA is shown using the following table:

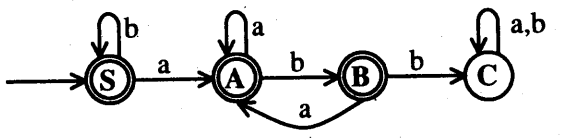
Transitions	Productions
δ(S, a) = A	S → aA
δ(S, b) = C	S → bC
δ(A, a) = C	A → aC
δ(A, b) = B	A → bB
δ(B, a) = B	B → aB
δ(B, b) = B	B → bB
δ(C, a) = C	C → aC
δ(C, b) = C	C → bC

It is very important to note that B is the final state. So, we have to introduce the production B → ε. The grammar G corresponding to the productions obtained is shown below:

G = (V, T, P, S) where

V = {S, A, B, C}
T = {a, b}
P = {
S → aA | bC
A → aC | bB
B → aB | bB | ε
C → aC | bC
}
S is the start symbol

Example 3.33: Construct a regular grammar for the following FA



Note that for each transition of the form δ(A, a) = B, introduce the production A → aB. If q ∈ F i.e., if q is the final state, introduce the production q → ε. The productions obtained from the transitions defined for FA is shown using the following table:

Transitions	Productions
$\delta(S, a) = A$	$S \rightarrow aA$
$\delta(S, b) = S$	$S \rightarrow bS$
$\delta(A, a) = A$	$A \rightarrow aA$
$\delta(A, b) = B$	$A \rightarrow bB$
$\delta(B, a) = A$	$B \rightarrow aA$
$\delta(B, b) = C$	$B \rightarrow bC$
$\delta(C, a) = C$	$C \rightarrow aC$
$\delta(C, b) = C$	$C \rightarrow bC$

It is very important to note that S, A and B are final states. So, we have to introduce the productions $S \rightarrow \varepsilon$, $A \rightarrow \varepsilon$ and $B \rightarrow \varepsilon$. The grammar G corresponding to the productions obtained is shown below:

Grammar G = (V, T, P, S) where
V = {S, A, B, C}
T = {a, b}
P = {

$$\begin{aligned} S &\rightarrow aA \mid bS \mid \varepsilon \\ A &\rightarrow aA \mid bB \mid \varepsilon \\ B &\rightarrow aA \mid bC \mid \varepsilon \\ C &\rightarrow aC \mid bC \end{aligned}$$

}
S is the start symbol

Note: The FA in this problem accepts strings of a's and b's except those containing the substring abb. So, from the grammar G we can obtain a regular language which consists of strings of a's and b's without the substring abb.

Example 3.34: Obtain a right linear grammar for the language $L = \{a^n b^m \mid n \geq 2, m \geq 3\}$

Note: In a right linear grammar either we should have purely terminals (including ε) or terminals followed by a single non-terminal.

It is clear from the given language that the language should start with at least two *a*'s which can be done using the production

$$S \rightarrow aaA$$

After applying this production, we have two *a*'s followed by A from which any number of *a*'s or at least three *b*'s can be generated. The corresponding productions are

A $\rightarrow$ aA | bbbB

Now, from B any number of *b*'s can be generated using the production

B $\rightarrow$ bB | ε

So, the final right linear grammar is G = (V, T, P, S) where

V = {S, A, B}
T = {a, b}
P = {

S $\rightarrow$ aaA
A $\rightarrow$ aA | bbbB
B $\rightarrow$ bB | ε

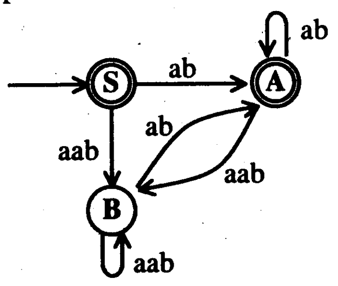
}
S is the start symbol

Note: Easiest way of solving this problem is to construct a DFA. Once we have a DFA, we can easily obtain the right linear grammar as we have done in examples 3.32 and 3.23.

Example 3.35: Obtain a right linear grammar for the regular expression ((aab)* ab)*

Note: Easiest way of solving this problem is to construct a DFA. Once we have a DFA, we can easily obtain the right linear grammar as we have done in examples 3.32 and 3.23.

The DFA for this regular expression is shown



Now, obtain the right linear grammar from the DFA as discussed in the pervious examples. The final right linear grammar is G = (V, T, P, S) where

V = {S, A, B}
T = {a, b}
P = {

S	$\rightarrow$	abA| aabB | ε
A	$\rightarrow$	abA | aabB| ε
B	$\rightarrow$	aabB| abA

}
S is the start symbol

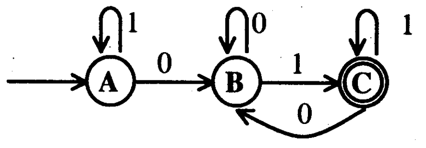
3.14 Left Linear Grammar from Finite Automaton

The left linear grammar can be obtained from FA as shown below:

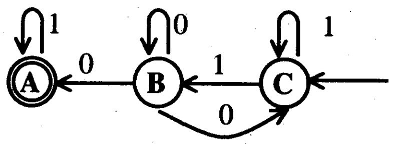
- Obtain the reverse of given DFA
- Obtain the right linear grammar from the reversed DFA
- Obtain the left linear grammar from right linear grammar.

Let us take some examples and see how a left linear grammar can be obtained.

Example 3.36: Obtain a left linear grammar for the DFA shown below



Step 1: Reverse the DFA i.e., make A as the final state and C as the start state and reverse the direction of the arrow. The reversed DFA is shown below:



Step 2: Obtain the right linear grammar for the above DFA. The corresponding productions are shown below:

C	$\rightarrow$	1C | 1B
B	$\rightarrow$	0B | 0A | 0C
A	$\rightarrow$	1A | ε

Step 3: Reverse the productions of right linear grammar to get left linear grammar. For example, if

A $\rightarrow$ abcdB

is the production in right linear grammar, after reversing the production will be of the form

$$A \rightarrow Bdcba$$

The conversion of right linear grammar to the left linear grammar is shown below:

Right linear grammar	Left Linear grammar
$C \rightarrow 1C \mid 1B$	$C \rightarrow C1 \mid B1$
$B \rightarrow 0B \mid 0A \mid 0C$	$B \rightarrow B0 \mid A0 \mid C0$
$A \rightarrow 1A \mid \varepsilon$	$A \rightarrow A1 \mid \varepsilon$

So, the final left linear grammar is

$V = \{C, A, B\}$
$T = \{0, 1\}$
$P = \{$

$$C \rightarrow C1 \mid B1$$
$$B \rightarrow B0 \mid A0 \mid C0$$
$$A \rightarrow A1 \mid \varepsilon$$

$\}$
$S = C$ is the start symbol

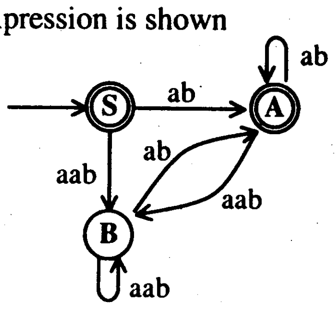
Now, whatever language is accepted by original DFA, the same language should be obtained from the left linear grammar. The string 10101 is accepted DFA. The same string can be derived from left linear grammar as shown below:

$$\begin{aligned}
C &\Rightarrow B1 && \text{(By applying } C \rightarrow B1\text{)}\\
&\Rightarrow A01 && \text{(By applying } B \rightarrow A0\text{)}\\
&\Rightarrow A101 && \text{(By applying } A \rightarrow A1\text{)}\\
&\Rightarrow A0101 && \text{(By applying } A \rightarrow A0\text{)}\\
&\Rightarrow A10101 && \text{(By applying } A \rightarrow A1\text{)}\\
&\Rightarrow 10101 && \text{(By applying } A \rightarrow \varepsilon\text{)}
\end{aligned}$$

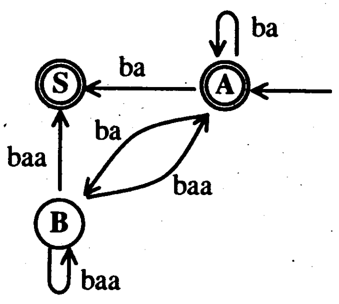
Hence, the left linear grammar obtained is equivalent to the given FA.

Example 3.37: Obtain a left linear grammar for the regular expression $((aab)^* ab)^*$

The DFA for this regular expression is shown



Step 1: Reverse the DFA. In the reversed DFA, A will be the start state and A and S will be the final states. After reversing the symbols and the arcs the corresponding DFA obtained is:



Step 2: Obtain the right linear grammar for the above DFA. The corresponding productions are shown below:

A → baA| baB|baS|ε
B → baaB| baaA| baaS
S → ε

Step 3: Reverse the productions of right linear grammar to get left linear grammar. For example, if

A → abcdB

is the production in right linear grammar, after reversing the production will be of the form

A → Bdcba

The conversion of right linear grammar to the left linear grammar is shown below:

Right linear grammar			Left Linear grammar		
A	→	baA| baB|baS|ε	A	→	Aab| Bab |Sab |ε
B	→	baaB| baaA| baaS	B	→	Baab |Aaab |Saab
S	→	ε	S	→	ε

V = {A, B, S}
T = {a, b}
P = {

$$A \rightarrow Aab \mid Bab \mid Sab \mid \varepsilon$$
$$B \rightarrow Baab \mid Aaab \mid Saab$$
$$S \rightarrow \varepsilon$$

}
A is the start symbol

Now, whatever language is accepted by original DFA, the same language should be obtained from the left linear grammar. The string aababaabab is accepted DFA. The same string can be derived from left linear grammar as shown below:

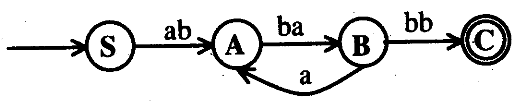
$A \Rightarrow Bab$	(By applying $A \rightarrow Bab$)
$\Rightarrow Aaabab$	(By applying $B \rightarrow Aaab$)
$\Rightarrow Aabaabab$	(By applying $A \rightarrow Aab$)
$\Rightarrow Saababaabab$	(By applying $A \rightarrow Saab$)
$\Rightarrow aababaabab$	(By applying $S \rightarrow \varepsilon$)

Hence, the left linear grammar obtained is equivalent to the given FA.

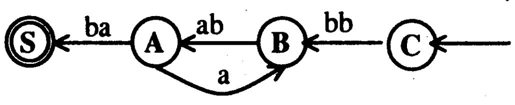
Example 3.38: Obtain a left linear grammar for right linear grammar shown below:

$$\mathbf{S} \rightarrow \mathbf{abA}$$
$$\mathbf{A} \rightarrow \mathbf{baB}$$
$$\mathbf{B} \rightarrow \mathbf{aA} \mid \mathbf{bb}$$

The DFA for the right linear grammar is shown below:



Step 1: Reverse the DFA i.e., make S as the final state and C as the start state and reverse the direction of the arrow. The reversed DFA is shown below:



Step 2: Obtain the right linear grammar for the above DFA. The corresponding productions are shown below:

$$C \rightarrow bbB$$
$$B \rightarrow abA$$
$$A \rightarrow ba \mid aB$$

Step 3: Reverse the productions of right linear grammar to get left linear grammar. For example, if

$$A \rightarrow abcdB$$

is the production in right linear grammar, after reversing the production will be of the form

$$A \rightarrow Bdcba$$

The conversion of right linear grammar to the left linear grammar is shown below:

Right linear grammar	Left Linear grammar
$C \rightarrow bbB$	$C \rightarrow Bbb$
$B \rightarrow abA$	$B \rightarrow Aba$
$A \rightarrow ba \mid aB$	$A \rightarrow ab \mid Ba$

So, the final left linear grammar is

V = {C, A, B}
T = {a, b}
P = {

$$C \rightarrow Bbb$$
$$B \rightarrow Aba$$
$$A \rightarrow ab \mid Ba$$

}
S = C is the start symbol

Now, whatever language is accepted by original DFA, the same language should be obtained from the left linear grammar. It is left to the reader to show the sample example so that the same language is accepted by right linear and left linear grammar. Hence, the left linear grammar obtained is equivalent to the given FA.

3.15 Regular Expressions in UNIX

A regular expression is a set of characters that specify a pattern. They are used to search for specific lines of text containing a particular pattern. The term "regular" is used to describe formal languages. Majority of the UNIX utilities operate on ASCII files one line at a time. The regular expressions will be used in majority of the editors to search for a specific pattern. The notations that are used in UNIX platform are shown below:

.	A dot will match any single character except a newline character.
*,+	Star and plus are used to match zero/one or more of the preceding expressions.
?	Matches zero or one copy of the preceding expression.
|	A logical 'or' statement - matches either the pattern before it, or the pattern after.
^	Matches the very beginning of a line.
$	Matched the end of a line.
/	Matches the preceding regular expression, but only if followed by the subsequent expression.
[ ]	Brackets are used to denote a character class, which matches any single character within the brackets. If the first character is a '^', this negates the brackets causing them to match any character except those listed. The '-' can be used in a set of brackets to denote a range. C escape sequences must use a '\'.
" "	Match everything within the quotes literally
( )	Group everything in the parentheses as a single unit for the rest of the expression.

The various regular expressions are described below:

[0-9]	A single digit
[0-9]+	An unsigned positive integer
[\t\n]+	Match white spaces such as tab, newline and space
[aeiouAEIOU]	Match the vowels
[a-zA-Z]	Match the consonants
"+"?[0-9]+	Match any positive integer. The operator "+" preceding the symbol "?" indicates that "+" is optional.
-[0-9]+	Match any negative integer

Using the shorthand notations, the regular expressions can be built. For example, consider the definition of regular expression as shown below:

```
exp ([eE]([-+])?[0-9]+)?
```

Here, "exp" is the shorthand name for the regular expression

```
([eE]([-+])?[0-9]+)?
```

describing the exponent part of the fraction. The symbol "?" indicates that it is the optional part. Using the above shorthand notation, we can write the regular expression to identify the positive fraction as shown below:

```
"+"?[0-9]*"."[0-9]+{exp}
```

Similarly, the regular expression to identify a negative fraction can be written as

```
-[0-9]*"."[0-9]+{exp}
```

Without using the shorthand notation, the regular expressions to identify a positive fraction and a negative fraction can be written as shown below:

```
"+"?[0-9]*"."[0-9]+([eE]([-+])?[0-9]+)?          /* Identify a positive fraction
*/
-[0-9]*"."[0-9]+([eE]([-+])?[0-9]+)?          /* Identify a negative fraction */
```

We know that an identifier starts with a letter from "a" through "z" or "A" through "Z" and followed by any combination of letters or digits and can be represented using regular expression as shown below:

```
[a-zA-Z]([a-zA-Z]|[0-9])*
```

The character sets can be combined by placing them next to each other. For example, if it is required to search for a word that

- Started with a capital letter "S"
- Was the first word on a line
- The second letter was a lower case letter
- Was exactly three letters long, and
- The third letter was a vowel

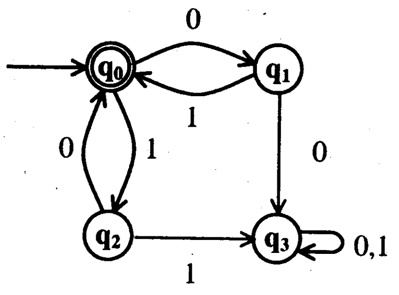
the regular expression would be "^S[a-z][aeiou]". The following table gives the regular expressions and equivalent UNIX notation for the regular expressions.

Regular expressions	UNIX Notation
(a+b)*	[ab]*
(a+b)*abb	[ab]*"abb"
ab(a+b)*	"ab"[ab]*
(a+b)*aa(a+b)*	[ab]*"aa"[ab]*
a*b*c*	[a]*[b]*[c]*
$a^+b^+c^+$	[a]+[b]+[c]+
aa*bb*cc*	[a][a]*[b][b]*[c][c]*
(a+b)* (a + bb)	[ab]*[a(bb)]
(aa)*(bb)*b	[(aa)]*[(bb)*]"b"
(0+1)*000	[01]*"000"
(11)*	[(11)*]
01* + 1	"0"[1]*|1

Exercises

1. Obtain the regular expressions for the following assuming $\Sigma = \{0, 1\}$
   a. String containing even number of 0's
   b. String not ending with 001
   c. string containing at least one 0
   d. String containing exactly one 0
   e. String containing not more than 3 0's
2. Obtain the regular expressions for the following assuming $\Sigma = \{0, 1\}$
   a. $L = \{a^n b^m c^p \mid n \leq 4, m \geq 2, p \leq 2\}$ Ans: $(\varepsilon + a + aa + aaa + aaaa)(bbb^*)(\varepsilon + c + cc)$
   b. $L = \{a^{2n} b^{2m+1} \mid n \geq 0, m \geq 0\}$ Ans: (aa)*b(bb)*
   c. $L = \{w : |w| \bmod 3 = 0\}$ Ans: ((a+b) (a+b) (a+b))*
3. Find DFAs to accept the following lanaguages
   a. L(00* + 010* 01)
   b. L(0(0+1)*11)
   c. L(ab(a+ab)* (a+aa))
4. Construct an NFA to accept to accept the regular expression (0+1)*(00+11)(0+1)*
5. Construct an NFA for the regular expression 10 + (0 + 11) 0*1 and obtain the corresponding DFA. Also reduce the states of DFA.

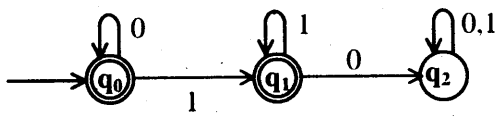
6. Obtain right linear grammar for the DFA obtained in exercise 4.
7. Obtain the right linear grammars for the DFAs obtained in exercise 2.
8. What does the following regular expression imply
    a. a(a+b)*ab
    b. (0*1 + 1*0)*0
    c. 0* + (01+0)*
9. Obtain the transition diagrams for the regular expression shown in exercise 7.
10. Obtain a DFA for the grammar $S \rightarrow 0S \mid 1S \mid 0A$, $A \rightarrow 1B$, $B \rightarrow 0C$, $C \rightarrow \varepsilon$.
11. For the DFA obtained in exercise 9 obtain the corresponding left linear grammar.
12. Construct a finite automaton to recognize the language generated by the grammar $S \rightarrow 0S \mid 1A \mid 1$, $A \rightarrow 0A \mid 1S0 \mid 1S \mid 0 \mid 1$
12. What is a regular expression? Give an example
13. Explain what each of the regular expressions represent in English
    a. (a+b)*aa(a+b)*
    b. a*b*c*
    c. $a^+b^+c^+$
    d. aa*bb*cc*
    e. a+b)* (a + bb)
    f. (aa)*(bb)*b
    g. (0+1)*000
14. Obtain a regular expression to accept a language consisting of strings of a's and b's of even length
15. Obtain a regular expression to accept a language consisting of strings of a's and b's of odd length
16. Obtain a regular expression such that L(R) = $\{w \mid w \in \{0, 1\}^*$ with at least three consecutive 0's
17. Obtain a regular expression to accept strings of a's and b's ending with 'b' and has no sub string aa
18. Obtain a regular expression for the FA shown below:



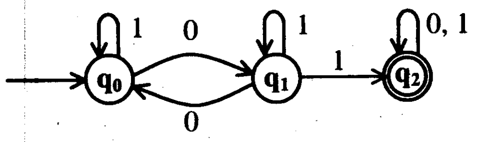
19. Obtain a regular expression to accept strings of 0's and 1's having no two consecutive zeros

20. Obtain a regular expression to accept strings of a's and b's of length ≤ 10
21. How an NFA can be obtained from R.E? Give the procedure.
22. Obtain an NFA which accepts strings of a's and b's starting with the string ab
23. Obtain an NFA for the regular expression a* + b* + c*
24. Obtain an NFA for the regular expression (a+b)*aa(a+b)*

25. What is the language accepted by the following FA



26. Obtain a regular expression for the FA shown below:



27. What are the applications of Regular expression
28. Obtain a regular expression to identify an identifier
29. Obtain a regular expression to identify an integer
30. Explain Chomsky Hierarchy of classification of grammars
31. What is an unrestricted grammar (type 0 grammar)? Give example.
32. What is context sensitive (type 1) grammar? Give example.
33. What is context free (type 2) grammar? Give example.
34. What is regular (type 3) grammar? Give example.
35. What is right linear grammar? Explain with example
36. What is left linear grammar? Explain with example
37. What is linear grammar? Explain with example
38. Prove that for every regular language generated by regular grammar there exists a corresponding DFA?
39. Construct a DFA to accept the language generated by the following grammar.

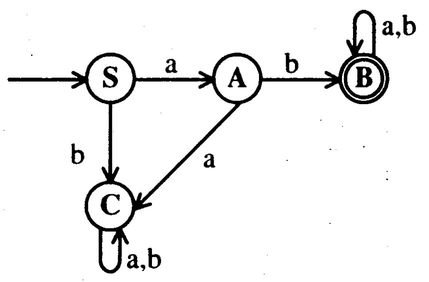
S	$\rightarrow$	01A
A	$\rightarrow$	10B
B	$\rightarrow$	0A|11

40. Construct a DFA to accept the language generated by the following grammar.

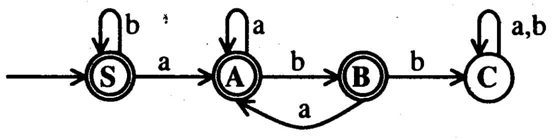
S	$\rightarrow$	aA | ε
A	$\rightarrow$	aA | bB | ε
B	$\rightarrow$	bB | ε

41. Prove that for every DFA, there exists a right linear grammar.

42. Construct a regular grammar from the following FA



43. Construct a regular grammar for the following FA

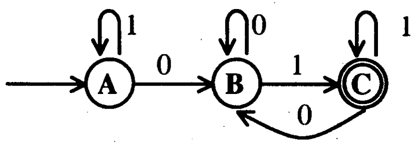


44. Obtain a right linear grammar for the language L = $\{a^n b^m \mid n \geq 2, m \geq 3\}$

45. Obtain a right linear grammar for the regular expression $((aab)^* ab)^*$

46. How a left linear grammar can be obtained from FA?

47. Obtain a left linear grammar for the DFA shown below



48. Obtain a left linear grammar for the regular expression $((aab)^* ab)^*$

49. Obtain a left linear grammar for right linear grammar shown below:

$$
\begin{aligned}
S &\rightarrow abA \\
A &\rightarrow baB \\
B &\rightarrow aA \mid bb
\end{aligned}
$$

4

Properties of Regular Languages

What we will know after reading this chapter?

- Properties of regular languages such as closure under
  - Union
  - Concatenation
  - Star
  - Complementation
  - Intersection
  - Difference
  - Homomorphism
- Proof for all the above properties
- Limitations of Regular languages
- Non regular languages
- Pigeonhole principle
- Pumping Lemma for regular languages
- Proof of Pumping Lemma
- Applications of Pumping Lemma
- General method of showing that certain languages are non-regular using Pumping Lemma
- Number of problems showing that the languages are non-regular using Pumping Lemma
- Reducing the states of DFA
- What are distinguishable and non-distinguishable states
- Solution to varieties of problems while reducing the states of DFA
- Solution to more than 20 problems of various nature

This chapter deals with closure properties of regular languages. The different closure properties covered in this chapter are union, concatenation, star-closure, intersection, complementation etc. This also covers pumping lemma which is a very useful concept to check whether the given language is regular or not.

4.1 Closure under Union, concatenation and star

Theorem: If L_1 and L_2 are regular, then L_1 U L_2, $L_1.L_2$ and L_1* also denote the regular languages and we say that the regular languages are closed under union, concatenation, start-closure.

Proof: Let L_1 and L_2 are regular languages corresponding to the regular expressions R_1 and R_2. By definition, $R_1 + R_2$, $R_1.R_2$ and R_1* are regular expressions and so L_1 U L_2, $L_1.L_2$, L_1* denote the regular languages and so regular languages are closed under union, concatenation and star-closure.

4.2 Closure under complementation

Theorem: If L_1 and L_2 are regular, then the regular language is closed under complementation.

Proof: Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA which accepts the language L_1. Now, let us define the machine $M_1 = (Q, \Sigma, \delta, q_0, Q\text{-}F)$. Note that there is no difference between M and M_1 except the final states. The non-final states of M are the final states of M_1 and final states of M are the non-final states of M_1. So, the language which is rejected by M is accepted by M_1. Also, a language accepted by a DFA is regular. So, the language accepted by M_1 is regular. So, a regular language is closed under complementation.

4.3 Closure under intersection

Theorem: If L_1 and L_2 are regular, then the regular language is closed under intersection.

Let us consider $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ which accepts L_1 and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ which accepts L_2. It is clear from these two machines that the alphabets of both machines are same. Let us assume both the machines are DFAs. To accept the language $L_1 \cap L_2$, let us construct the machine M that simulates both M_1 and M_2 where the states of the machine M are the pairs (p, q) where $p \in Q_1$ and $q \in Q_2$. The transition for the machine M from the state (p, q) on input symbol $a \in \Sigma$ is the $(\delta_1(p, a), \delta_2(q, a))$ i.e., if $\delta_1(p, a) = r$ and $\delta_2(q, a) = s$, then the machine moves from the state (p, q) to the state (r, s) on input symbol *a*. In this manner, the machine M can simulate the effect of M_1 and M_2. Now, the machine $M = (Q, \Sigma, \delta, q, F)$ recognizes $L_1 \cap L_2$ where

$Q = Q_1 \times Q_2$

$q = (q_1, q_2)$ where q_1 and q_2 are the start states of machine M_1 and M_2 respectively.

$F = \{ (p, q) \mid p \in F_1 \text{ and } q \in F_2 \}$

δ: $Q \times \Sigma$ to Q is defined by $\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$

It is clear from δ that

$\hat{\delta}$ ((p, q), w) = ($\hat{\delta}_1$(p, w), $\hat{\delta}_2$(q, w))

and the string w is accepted only if

$\hat{\delta}$ ((q_1, q_2), w) $\in$ F

which implies ($\hat{\delta}_1$(q_1, w), $\hat{\delta}_2$(q_2, w)) $\in$ F. This will happen if and only if

$\hat{\delta}_1$(q_1, w) $\in F_1$ and $\hat{\delta}_2$(q_2, w) $\in F_2$

i.e., if and only if w $\in L_1 \cap L_2$. So, the regular language is closed under intersection.

4.4 Closure under difference

Theorem: If L_1 and L_2 are regular, then the regular language is closed under difference.

Let us consider $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ which accepts L_1 and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ which accepts L_2. We define M = (Q, Σ, δ, q, F) recognizing $L_1 - L_2$ as follows.

$Q = Q_1 \times Q_2$

$q = (q_1, q_2)$

$F = \{ (p, q) \mid p \in F_1 \text{ and } q \notin F_2 \}$

δ: Q x Σ to Q is defined by δ ((p, q), a) = (δ_1(p, a), δ_2(q, a))

It is clear from δ that

$\hat{\delta}$ ((p, q), w) = ($\hat{\delta}_1$(p, w), $\hat{\delta}_2$(q, w))

and the string w is accepted only if

δ ((q_1, q_2), w) $\in$ F

which implies ($\hat{\delta}_1$(q_1, w), $\hat{\delta}_2$(q_2, w)) $\in$ F. This will happen if and only if

$\hat{\delta}_1$(q_1, w) $\in F_1$ and $\hat{\delta}_2$(q_2, w) $\notin F_2$

i.e., if and only if w $\in L_1 - L_2$. So, the regular language is closed under difference.

4.5 Closure under homomorphism

Definition: Let Σ and Γ are set of alphabets. The function

$$h: \Sigma \rightarrow \Gamma^*$$

is called homomorphism i.e., a substitution where a single letter is replaced by a string. If

$$w = a_1a_2a_3....a_n,$$

then

$$h(w) = h(a_1)h(a_2).....h(a_n)$$

If L is made of alphabets from Σ, then $h(L) = \{h(w) \mid w \in L\}$ is called homomorphic image.

Example 4.1: Let $\Sigma = \{0, 1\}$ $\Gamma = \{0, 1, 2\}$ and h(0) = 01, h(1) = 112. What is h(010)? If L = {00, 010} what is homormorphic image of L?

By definition we have

$$h(w) = h(a_1)h(a_2)....h(a_n)$$

So,

$$\begin{aligned} h(010) &= h(0)h(1)h(0) \\ &= 0111201 \end{aligned}$$

$$\begin{aligned} L\,(00, 010\} &= L(\,h(00), h(010)) \\ &= L(\,h(0)h(0), h(0)h(1)h(0)) \\ &= L(\,0101, 0111201) \end{aligned}$$

Therefore,

$$\begin{aligned} h(010) &= 0111201 \\ L(00, 010) &= L(\,0101, 0111201) \end{aligned}$$

Example 4.2: If $\Sigma = \{0, 1\}$ $\Gamma = \{1, 2, 3\}$, h(0) = 3122, h(1) = 132, what is (0+1)*(00)*?

By definition we have

$$h(w) = h(a_1)h(a_2)....h(a_n)$$

So,

$$\begin{aligned} (0+1)^*(00)^* &= (h(0) + h(1))^*\ (h(0)h(0))^* \\ &= (3122 + 132)^*\ (31223122)^* \end{aligned}$$

Theorem: If L is regular and h is homomorphism, then homomorphic image h(L) is regular.

Proof: Let R be the regular expression and L(R) be the corresponding regular language. We can easily find h(R) by substituting h(a) for each a in Σ. By definition of regular expression, h(R) is a regular expression and so h(L) is regular language. So, the regular language is closed under homomorphism.

4.6 Limitations of Regular Languages

So far we have spent most of our time on regular languages, discussing the different ways to specify them, the various operations performed on them and their closure properties etc. Since some restrictions are there in regular grammars, the regular language should be a subset of all possible languages. It means that some are regular languages and some are not.

Then, what type of languages are regular and what are not regular? The answer is: every finite language is regular. Any finite language can be expressed using regular expression and for any language which can be represented using regular expression, we can have a DFA. Even some of the infinite languages are also can be represented using Regular expressions and for these languages also we can construct DFA and so are regular.

But, some of the infinite languages are not regular. Though the regular languages are important, there are non-regular languages which are very interesting and important. Some of the non-regular languages are:

1. $\{w \in \{0, 1\} \mid w \text{ contains an equal number of 0s and 1s }\}$
2. $\{0^n1^n \in \{0, 1\}^* \mid n \geq 0\}$
3. $\{a^p \in \{a\}^* \mid p \geq 2 \text{ is a prime number }\}$
4. Check for the matching parentheses (not possible using DFA)
5. Count number of a's and then the number of b's (not possible using DFA) and so it can not be used as a counter

It is clear from these examples that it is not possible to obtain corresponding DFA for these and so they are not regular. So, we are facing problem whenever we encounter non regular languages. The easiest way to prove that a language is regular is to construct a DFA. Then, what is the easiest way to prove that a language is not regular? The answer for this is the pumping lemma.

4.7 Non Regular Languages

In this section, we discuss the way to prove that certain languages are infinite using Pumping Lemma. The principle used in Pumping Lemma is similar to the Pigeonhole principle.

Pigeonhole principle: The pigeonhole principle is based on a simple observation. Suppose there are *n* objects and *m* boxes where number of objects *n*, are greater than number of boxes *m*. In this case, if all *n* objects are placed into *m* boxes, then at least one box will have more than one object.

The pumping lemma is based on this principle. The advantages of pumping lemma are:

1. We can easily prove that certain languages are nonregular.
2. It is possible to check whether a language accepted by FA is finite or infinite.

The pumping lemma is stated as follows:

Pumping Lemma: Let M = (Q, Σ, δ, q_0, F) be an FA and has *n* number of states. Let L be the language accepted by M and assume the language is regular. Let $x \in L$ and $|x| \geq n$ i.e., length of the string x is greater than the number of states of FA. If the string x can be decomposed into combinations of strings

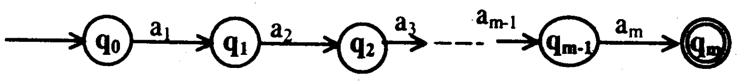
$$x = uvw$$

such that $|uv| \leq n$, $|v| \geq 1$, then $uv^iw \in L$ for i = 0, 1, 2,......

Proof: Let M = (Q, Σ, δ, q_0, F) be an FA and Let $x = a_1a_2a_3...a_m$ is the input which is accepted by the machine. Assume the machine has *n* states and assume that $m \geq n$.

Note: If exactly one symbol is accepted by an FA then there are two distinct states in it. Similarly to accept exactly two symbols, FA should have three distinct states. In general, to accept a string x where $|x| = n$, then the FA should have n+1 states. Let those states be $q_0, q_1, q_2......q_n$.

Since we have m input symbols, naturally we should have m+1 states in the sequence $q_0, q_1, q_2......q_m$, where q_0 will be the start state and q_m will be the final state as shown below:



It is clear from the figure that $\delta(q_{i-1}, a_i) = q_i$ for each $1 \leq i \leq m$. In this chain of states from q_0 to q_m, suppose a state *q* appears more than once, say $q = q_i$ and $q = q_j$ where $i < j$. In this case, the chain should have a loop and we can split into three groups:

1. The first group is the string prefix from $a_1a_2...a_i$
2. The second group is the loop string from $a_{i+1}a_{i+2}.....a_{j-1}a_j$
3. The third group is the string suffix from $a_{j+1}a_{j+2}...a_m$

as show in figure 4.1.

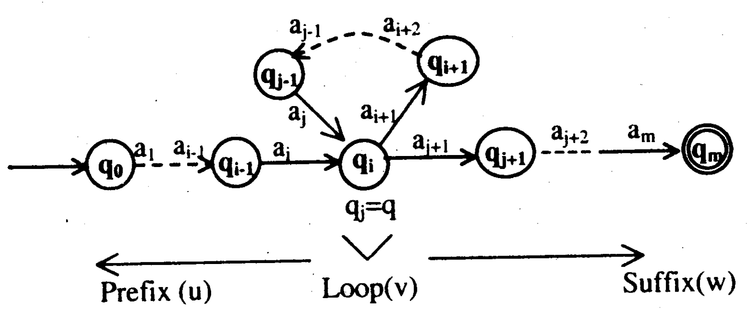


Fig. 4.1 DFA to explain Pumping Lemma

Note that the machine cannot remember the previous state or it does not know how it has reached the current state. So, whenever we say that the machine has reached the state q, the machine does not know whether the state is reached after the end of the prefix or after it has processed the loop. Since the machine cannot distinguish these states, after the prefix (u), the machine may be in a loop for zero or more time (accepting the string v zero or more times) and then accept the suffix string (w). In general, if the string x is split into sub strings uvw, then for all $i \geq 0$,

$$uv^iw \in L$$

If $F = \{q_m\}$, $a_1a_2 \ldots\ldots a_m \in L(M)$, then

$$a_1a_2 \ldots\ldots a_{i-1}a_ia_{i+1}a_{i+2} \ldots\ldots a_{j-2}a_{j-1}a_ja_{j+1}a_{j+2} \ldots a_m \in L(M).$$

This can be expressed using the transitions as shown below:

$$\begin{aligned}
&\delta(q_0, a_1a_2 \ldots\ldots\ldots a_{i-1}a_ia_{j+1}a_{j+2} \ldots\ldots\ldots a_m) \\
&\quad = \delta(\,\delta(q_0, a_1a_2 \ldots\ldots\ldots a_{i-1}a_i), a_{j+1}a_{j+2} \ldots\ldots\ldots a_m) \\
&\quad = \delta(q, a_{j+1}a_{j+2} \ldots\ldots\ldots a_m) \\
&\quad = \delta(q_k, a_{k+1}a_{k+2} \ldots\ldots\ldots a_m) \\
&\quad = q_m
\end{aligned}$$

Also, after the string

$$a_1a_2 \ldots\ldots\ldots a_i$$

the machine will be in state q_i. Since q_i and q_j are same, we can input the string

$$a_{i+1}a_{i+2} \ldots\ldots\ldots a_j$$

any number of times and the machine will stay in q_j only. Finally, if the input string is

$$a_{j+1}a_{j+2} \ldots\ldots\ldots a_m$$

the machine enters into final state q_m.

4.8 Applications of Pumping Lemma

All languages are not regular. A nonregular language is that language for which a FA cannot be constructed. So, to prove that certain languages are not regular, we use pumping lemma. The typical applications of Pumping Lemma are:

1. To prove that certain languages are not regular. The pumping lemma *cannot* be used to prove that a given language *is* regular.
2. To check whether the language is infinite. If there is a string x such that $|x| \geq$ the number of states accepted by DFA M, then L(M) is infinite. Otherwise, L(M) is finite. So, using Pumping lemma we can check whether the language is finite or infinite.

The general strategy used to prove that certain language is not regular is shown below.

1. Assume that the language L is regular and the number of states in FA be n.
2. Select the string say x and break it into substrings u, v and w such that x = uvw with the constraints $|x| \geq n$, $|uv| \leq n$ and $|v| \geq 1$.
3. Find any i such that $uv^iw \notin L$. According to pumping lemma, $uv^iw \in L$. So, the result is contradiction to the assumption that the language is regular. Therefore, the given language L is not regular.

Example 4.3: Show that $L = \{ww^R \mid w \in (0+1)^*\}$ is not regular.

Step1: Let L is regular and *n* be the number of states in FA. Consider the string

$$x = \overbrace{\underbrace{1\ldots1}_{n}\ \underbrace{0\ldots0}_{n}}^{w}\ \overbrace{\underbrace{0\ldots0}_{n}\ \underbrace{1\ldots1}_{n}}^{w^R}$$

where n is the number of states of FA, $w = 1\ldots10\ldots0$ and reverse of w is given by
$w^R = 0\ldots01\ldots1$.

Step 2: Since $|x| \geq n$, we can split the string x into uvw such that $|uv| \leq n$ and $|v| \geq 1$ as shown below.

$$x = \underbrace{1\ldots\ldots}_{u}\ \underbrace{1}_{v}\ \underbrace{0\ldots0\ 0\ldots\ldots0\ 1\ldots\ldots1}_{w}$$

where $|u| = n-1$ and $|v| = 1$ so that $|uv| = |u| + |v| = n-1 + 1 = n$ which is true. According to pumping lemma, $uv^iw \in L$ for i = 0, 1, 2,...

Step 3: If i is 0 i.e., v does not appear and so the number of 1's on the left of x will be less than the number of 1's on the right of x and so the string is not in the form ww^R. So, $uv^iw \notin L$ when i = 0. This is a contradiction to the assumption that the language is regular.

So, the language $L = \{ww^R \mid w \in \{0+1\}^*\}$ is not regular.

Example 4.4: Show that $L = \{a^nb^n \mid n \geq 0\}$ is not regular.

Step1: Let L is regular and *n* be the number of states in FA. Consider the string $x = a^nb^n$.

Step2: Note that $|x| = 2n$ and is greater than n. So, we can split x into uvw such that $|uv| \leq n$ and $|v| \geq 1$ as shown below.

$$x = \overbrace{\underbrace{aaaaaa}_{u}\underbrace{a}_{v}}^{n}\overbrace{\underbrace{bbbbbbb}_{w}}^{n}$$

where $|u| = n-1$ and $|v| = 1$ so that $|uv| = |u| + |v| = n-1 + 1 = n$ and $|w| = n$. According to pumping lemma, $uv^iw \in L$ for i = 0, 1, 2,...

Step 3: If i is 0 i.e., v does not appear and so the number of a's will be less than the number of b's and so the string x does not contain some number of a's followed by same number of b's (equal to that of a's). Similarly, if i = 2, 3, ..., then number of a's will be more than the number of b's and so number of a's followed by equal number of b's does not exist. But, according to pumping lemma, n number of a's should be followed by n number of b's which is a contradiction to the assumption that the language is regular.

So, the language $L = \{a^nb^n \mid n \geq 0\}$ is not regular.

Example 4.5: Show that $L = \{a^nb^l \mid n \neq l\}$ is not regular.

It is given that $L = \{a^nb^l \mid n \neq l\}$. Let L is regular and *n* be the number of states in FA. Since L is regular, $\overline{L}$ is also regular (Based on Closure property). We know that

$$\overline{L} \cap L(a^*b^*) = \{a^nb^n \mid n \geq 0\} = L_1$$

where L(a*b*) is regular. Since we have assumed that L is regular, $\overline{L}$ is also regular(based on the closure property). Since $\overline{L}$ and $L(a^*b^*)$ are regular the language

$$L_1 = \{a^n b^n \mid n \geq 0\}$$

is also regular. But, we have already proved in earlier example 4.4 that L_1 is not regular. It is contradiction to the assumption that the language L is regular. So, the language

$$L = \{ a^n b^l \mid n \neq l \}$$

is not regular.

Example 4.6: Show that $L = \{a^i b^j \mid i > j\}$ is not regular.

Step1: Let L is regular and *n* be the number of states in FA. Consider the string $x = a^{n+1}b^n$.

Step2: Note that $|x| = 2n+1 \geq n$. So, we can split x into uvw such that $|uv| \leq n$ and $|v| \geq 1$ as shown below.

$$x = a^{n+1}b^n = \underset{u}{a^j} \quad \underset{v}{a^k} \quad \underset{w}{ab^n}$$

where $|u| = j$ and $|v| = k \geq 1$ and so that $|uv| = |u| + |v| = j+k \leq n$.

Step 3: According to pumping lemma, $uv^iw \in L$ for $i = 0$
i.e., $a^j (a^k)^i ab^n \in L$ for $i = 0$

Now, if we choose $i = 0$, number of a's in string u will not be more than the number of b's in w which is contradiction to the assumption that number of a's are more than the number of b's.

So, the language $L = \{a^i b^j \mid i > j\}$ is not regular.

Example 4.7: Show that $L = \{a^n b^l c^{n+l} \mid n,l \geq 0\}$ is not regular.

Step1: Let L is regular and *n* be the number of states in FA.

Step2: Since L is regular, it is closed under homomorphism. So, we can take

$$h(a) = a,\ h(b) = a \text{ and } h(c) = c.$$

Now, the language L is reduced to

$$L = \{ a^n a^l c^{n+l} \mid n+l \geq 0 \}$$

which can be written as

$$L = \{ a^{n+l} c^{n+l} \mid n+l \geq 0 \}$$

which is of the form

$$L = \{a^i b^i \mid i \geq 0\}$$

We know that the above language is not regular (proved in example 4.4) which is contradiction to the assumption that the language is regular. So, the given language

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

is not regular.

Example 4.8: Show that $L = \{a^{n!} \mid n \geq 0\}$ is not regular.

Note: n! = 1*2*3*.....n

Step1: Let L is regular and *n* be the number of states in FA.

Step2: Let $x = a^{n!}$. It is clear that $|x| \geq n$. So, we can split x into uvw such that $|uv| \leq n$ and $|v| \geq 1$ as shown below.

$$x = a^{n!} = \underset{u}{a^j}\ \underset{v}{a^k}\ \underset{w}{a^{n!-j-k}}$$

where $|u| = j$ and $|v| = k \geq 1$ and so that $|uv| = |u| + |v| = j+k \leq n$.

Step 3: According to pumping lemma, $uv^iw \in L$ for $i = 0, 1, 2,\ldots$
i.e.,

$$a^j\,(a^k)^i\,a^{n!-j-k} \in L$$

So, if we choose i = 0, it means that

$$a^j\,a^{n!-j-k} \in L \text{ (i.e., } uw \in L)$$

$$\Downarrow \text{ (implies)}$$

$$a^{n!-k} \in L$$

It is very clear that $n! > n! - k$. Now, when $k = 1$,

$$n! > n!-1$$

But, according to Pumping Lemma $n! = n! - 1$ which is not and is a contradiction. So, L can not be regular.

So, the language $L = \{ a^{n!} \mid n \geq 0\}$ is not regular.

Example 4.9: Show that $L = \{0^n \mid n \text{ is prime}\}$ is not regular.

Note: The language generated from this can take the following form

$$L = \{00, 000, 00000, \ldots\ldots\ldots\ldots\}$$

Step1: Let L is regular and *n* be the number of states in FA. Let us choose the value of x which depends on n.

$$\text{Let } x = 0^n \in L \text{ where n is prime.}$$

Step2: Note that $|x| = n$ and so, we can split x into uvw such that $|uv| \le n$ and $|v| \ge 1$ as shown below.

$$x = 0^n = \underbrace{0^j}_{u}\ \underbrace{0^k}_{v}\ \underbrace{0^{n-j-k}}_{w}$$

where $|u| = j$ and $|v| = k \ge 1$ and so that $|uv| = |u| + |v| = j+k \le n$.

Step 3: According to pumping lemma, $uv^iw \in L$ for $i = 0, 1, 2, \ldots$

i.e.,

$$0^j (0^k)^i 0^{n-j-k} \in L$$

i.e.,

$$j + ki + n - j - k = n + k(i - 1) \text{ is prime for all } i \ge 0$$

Now, if we choose $i = n + 1$, then

$$n + k(i - 1) = n + kn = n(k+1)$$

is also a prime for each

$$k \ge 1$$

which is a contradiction (because if $k = 1$, it will not be a prime) to the assumption that the language is regular. So, the language $L = \{0^n \mid n \text{ is prime}\}$ is not regular.

Example 4.10: Show that $L = \{w \mid n_a(w) = n_b(w)\}$ is not regular. Is L^* regular?

Note: The language generated from this can take the following form

$$L = \{\varepsilon, ab, ba, abab, aabb, bbaa, \ldots\ldots\ldots\}$$

Step1: Let L is regular and *n* be the number of states in FA. Let us choose the value of x which depends on n.

$$\text{Let } x = a^n b^n \in L$$

Step2: Note that $|x| = 2n \geq n$ and so, we can split x into uvw such that $|uv| \leq n$ and $|v| \geq 1$ as shown below.

$$x = a^nb^n = \underbrace{a^j}_{u}\ \underbrace{a^k}_{v}\ \underbrace{b^n}_{w}$$

where $|u| = j$ and $|v| = k \geq 1$ and so that $|uv| = |u| + |v| = j+k \leq n$.

Step 3: According to pumping lemma, $uv^iw \in L$ for $i = 0, 1, 2, \ldots$
i.e.,

$$a^j\ (a^k)^i\ b^n \in L \text{ for } i = 0, 1, 2, \ldots$$

Now, if we choose $i = 0$, number of a's will be less than the number of b's and if we choose $i = 2$, uv^2w will have more a's than b's which is contradiction to the assumption that it has equal number of a's anb b's.

So, the language $L = \{\ w \mid n_a(w) = n_b(w)\}$ is not regular. Since L is not regular L* is also not regular.

Example 4.11: Show that $L = \{\ w \mid n_a(w) < n_b(w)\}$ is not regular.

Step1: Let L is regular and *n* be the number of states in FA. Let us choose the value of x which depends on n.

$$\text{Let } x = a^{n-1}b^n \in L$$

Step2: Note that $|x| = 2n - 1 \geq n$ and so, we can split x into uvw such that $|uv| \leq n$ and $|v| \geq 1$ as shown below.

$$x = a^{n-1}b^n = \underbrace{a^{n-1}}_{u}\ \underbrace{b^k}_{v}\ \underbrace{b^{n-k}}_{w}$$

where $|u| = n-1$ and $|v| = k$ and so that $|uv| = |u| + |v| = n-1 + k \leq n$. To satisfy this condition the value of k should be less than or equal to 1 (in this case).

Step 3: According to pumping lemma, $uv^iw \in L$ for $i = 0, 1, 2, \ldots$
i.e.,

$$a^{n-1}\ (b^k)^i\ b^{n-k} \in L \text{ for } i = 0, 1, 2, \ldots$$

Now, if we choose $i = 0$,

$$a^{n-1}\ b^{n-k} \in L$$

$$\Downarrow \text{(implies)}$$

$$a^{n-1}\ b^{n-1} \in L \text{ (when } k = 1)$$

So, when i = 0, the number of a's and b's are same, which is contradiction to the assumption that the number of a's will be less than the number of b's. So, the language $L = \{ w \mid n_a(w) < n_b(w)\}$ is not regular.

Example 4.12: Show that $L = \{ ww \mid w \in \{a,b\}^* \}$ is not regular.

Step1: Let L is regular and *n* be the number of states in FA. Let us choose the value of x which depends on n. Let $x = a^nba^nb \in L$

Step2: Note that $|x| = 2n + 2 \geq n$ and so, we can split x into uvw such that $|uv| \leq n$ and $|v| \geq 1$ as shown below.

$$x = a^nba^nb = \underset{u}{a^j}\ \underset{v}{a^k}\ \underset{w}{ba^nb}$$

where $|u| = j$ and $|v| = k \geq 1$ and so that $|uv| = |u| + |v| = j+k \leq n$.

Step 3: According to pumping lemma, $uv^iw \in L$ for i = 0, 1, 2,...
i.e.,

$$a^j (a^k)^i\ ba^nb \in L \text{ for } i = 0, 1, 2, \ldots$$

Now, if we choose i = 0, number of a's on the left of first b will be less than the number of a's after the first b which is contradiction to the assumption that they are not equal. So, the language $L = \{ ww \mid w \in \{a,b\}^* \}$ is not regular.

Example 4.13: Show that $L = \{ a^n \mid n = k^2 \text{ for } k \geq 0\}$ is not regular.

Note: The above language can also be defined as: $L = \{ a^n \mid n$ is a perfect square $\}$

Step1: Let L is regular and *n* be the number of states in FA. Let us choose the value of x which depends on n. Let $x = a^m \in L$ (where $m = n^2$)

Step2: Note that $|x| \geq n$ and so, we can split x into uvw such that $|uv| \leq n$ and $|v| \geq 1$ as shown below.

$$x = a^m = \underset{u}{a^j}\ \underset{v}{a^k}\ \underset{w}{a^{m-j-k}}$$

where $|u| = j$ and $|v| = k \geq 1$ and so that $|uv| = |u| + |v| = j+k \leq n$.

Step 3: According to pumping lemma, $uv^iw \in L$ for i = 0, 1, 2,...
i.e.,

$$a^j \quad a^{ki} \quad a^{m-j-k} \in L \text{ for } i = 0, 1, 2, \ldots$$

Now, if we choose i = 2, we have

$$uv^2w \in L$$

i.e.,

$$a^j a^{2k} a^{m-j-k} \in L$$

i.e.,

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

Since $k \geq 1$, we have,

$$|a^{m+k}| = m + k = n^2 + k \text{ (since } m = n^2)$$

Note that

$$n^2 < n^2 + k < n^2 + 1 \text{ (when } k = 1) < (n^2 + 2n + 1) = (n+1)^2$$

and so

$$n^2 < (n+1)^2$$

Since n^2+k (nothing but m+k) lies between n^2 and $(n+1)^2$, it is not a perfect square which is contradiction to the assumption that it should be a perfect square (According to Pumping lemma). So, the language $L = \{ a^n \mid n = k^2 \text{ for } k \geq 0\}$ is not regular.

4.9 Limitations of finite automaton

Note: There are so many problems for which we can not construct a DFA and still we want some solution to solve those problems. For example,

1. Check for the matching parentheses (not possible using DFA)
2. Count number of a's and then the number of b's (not possible using DFA) and so it can not be used as a counter

The limitations of finite automaton are:

1. An FA has finite number of states and so it does not have the capacity to remember arbitrary long amount of information
2. Since it does not have memory FA can not remember a long string. For example, to check for matching parentheses, check whether the string is a palindrome or not etc.
3. Finite automata or finite state machines have trouble recognizing various types of languages involving counting, calculating, storing the string.

The solution for all these problems is to have a machine which is more general than DFA which can recognize these set of languages. In the coming chapters we discuss how these problems can be solved using Grammars, push down automaton, Turing machines etc.

4.10 Equivalence of two states

The language generated by a DFA is unique. But, there can exist many DFA's that accept the same language. In such cases, the DFA's are said to be equivalent. During computations, it is desirable to represent the DFA with fewer states since the space is proportional to the number of states of DFA. For storage efficiency, it is required to reduce the number of states and there by minimize the DFA. This can be achieved first by finding the distinguishable and indistinguishable states. First, let us see the definitions of *distinguishable* and *indistinguishable* states.

Definition: Two states *p* and *q* of a DFA are *equivalent* (*indistinguishable*) if and only if $\delta(p, w)$ and $\delta(q, w)$ are final states or both $\delta(p, w)$ and $\delta(q, w)$ are non-final states for all $w \in \Sigma^*$ i.e. if

$$\delta(p, w) \in F \text{ and } \delta(q, w) \in F$$

then the states p and q are *indistinguishable*. If

$$\delta(p, w) \notin F \text{ and } \delta(q, w) \notin F$$

then also the states p and q are *indistinguishable*. If there is at least one string *w* such that one of

$$\delta(p, w) \text{ and } \delta(q, w)$$

is final state and the other is non-final state, then the states p and q are not equivalent and are called *distinguishable* states.

The *distinguishable* and *indistinguishable* states can be obtained using *table-filling algorithm* (also called *mark* procedure) recursively as shown below:

Note: Before applying this procedure, ensure that the all the states of a DFA are reachable from the start state and delete all those states that are not reachable from the start state.

Step 1:

For each pair (p, q) where $p \in Q$ and $q \in Q$, if $p \in F$ and $q \notin F$ or vice versa then, the pair (p, q) is distinguishable and mark the pair (p, q) [by putting say 'x' for the pair (p,q)]

Step 2:

For each pair (p, q) and for each $a \in \Sigma$ find $\delta(p, a) = r$ and $\delta(q, a) = s$. If the pair (r, s) is already marked as distinguishable (Note: The distinguished states are already marked as 'x' as the first step) then the pair (p, q) is also distinguishable and mark it as say 'x'. Repeat step 2 until no previously unmarked pairs are marked.

Note: If the pair (p, q) obtained using the above *table filling algorithm* (mark procedure) are indistinguishable then the two states *p* and *q* are equivalent.

Example 4.14: Obtain the distinguishable table for the automaton shown below.

	Σ	
States	0	1
→A	B	F
B	G	C
Ⓒ	A	C
D	C	G
E	H	F
F	C	G
G	G	E
H	G	C

Solution: First let us obtain the table for inequalities (distinguishable or indistinguishable). This we can do by knowing the various states of given DFA. The states of the DFA are:

A B C D E F G H

The pairs can be obtained as shown below: (This can be obtained by writing all the pairs in the matrix form and take only the upper triangular matrix to avoid duplicate elements and cycles)

(A, B)	(A,C)	(A,D)	(A,E)	(A,F)	(A,G)	(A,H)
	(B,C)	(B,D)	(B,E)	(B,F)	(B,G)	(B,H)
		(C,D)	(C,E)	(C,F)	(C,G)	(C,H)
			(D,E)	(D,F)	(D,G)	(D,H)
				(E,F)	(E,G)	(E,H)
					(F,G)	(F,H)
						(G,H)

The above pairs can be represented in the tabular form as shown below and note that no pairs are marked.

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

OR

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

Step 1:

For each pair (p, q) where $p \in Q$ and $q \in Q$, if $p \in F$ and $q \notin F$ or vice versa then, the pair (p, q) is distinguishable and mark the pair (p, q) [by putting say 'x' for the pair (p,q)]. This clearly indicates that one of the pair should be a final state and the other should be a non final state. So, the various pairs which satisfy the above criteria are all the pairs involving the final state C and are marked as shown below:

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

Step 2:

For each pair (p, q) and for each $a \in \Sigma$ find $\delta(p, a) = r$ and $\delta(q, a) = s$. If the pair (r, s) is already marked as distinguishable (Note: The distinguished states are already marked as 'x' in the first step) then the pair (p, q) is also distinguishable and mark it as say 'x'. Repeat step 2 until no previously unmarked pairs are marked

Now consider the pairs (p,q) which are not marked in step 1 and obtain the corresponding pairs (r, s) on input symbols 0 and 1 as shown in table 4.1.

Note: If the pairs (r, s) are marked in step 1, the corresponding pair (p, q) should be marked. For example, for the pair (A,B) on 1 is (F,C) and is marked as 'x' in step 1. So, mark the pair (A,B) (shown using shading). On similar lines identify the pairs (r, s) which are marked in step 1 (shown in bold phase) and take the corresponding pairs (p,q) (shown in shading) and update the marking table with 'x' as shown in the marking table 4.2.

δ	**0**	**1**
(p,q)	**(r,s)**	**(r,s)**
(A,B)	(B,G)	**(F,C)**
(A,D)	**(B,C)**	(F,G)
(A,E)	(B,H)	(F,F)
(A,F)	**(B,C)**	(F,G)
(A,G)	(B,G)	(F,E)
(A,H)	(B,G)	**(F,C)**
(B,D)	**(G,C)**	**(C,G)**
(B,E)	(G,H)	**(C,F)**
(B,F)	**(G,C)**	**(C,G)**
(B,G)	(G,G)	**(C,E)**
(B,H)	(G,G)	(C,C)
(D,E)	**(C,H)**	(G,F)
(D,F)	(C,C)	(G,G)
(D,G)	**(C,G)**	(G,E)
(D,H)	**(C,G)**	**(G,C)**
(E,F)	**(H,C)**	(F,G)
(E,G)	(H,G)	(F,E)
(E,H)	(H,G)	**(F,C)**
(F,G)	**(C,G)**	(G,E)
(F,H)	**(C,G)**	**(G,C)**
(G,H)	(G,G)	**(E,C)**

Table 4.1 transitions for each pair(p,q) on 0 and 1

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

Table 4.2 transitions for the transitions of table 3.1

Again consider the pairs (p,q) which are not marked in the above table and see whether the corresponding pairs (r, s) on 0 and 1 are marked as shown below:

	0	**1**
(p,q)	**(r,s)**	**(r,s)**
(A,E)	(B,H)	(F,F)
(A,G)	**(B,G)**	**(F,E)**
(B,H)	(G,G)	(C,C)
(D,F)	(C,C)	(G,G)
(E,G)	**(H,G)**	**(F,E)**

Mark the pairs (A.G) and (E,G) with 'x' as shown below:

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

Observe that the pairs (A,E), (B,H) and (D,F) are not at all marked and are considered to be indistinguishable states. The other states namely C and G are distinguishable states.

4.11 Minimization of DFA

Once we have found the distinguishable and indistinguishable pairs we can easily minimize the number of states of a DFA and accepting the same language as accepted by original DFA. The algorithm to minimize the DFA is shown below:

Stepl: Find the distinguishable and indistinguishable pairs using the *table-filling* algorithm (discussed in previous section)

Step2: Partition the states Q into various groups. These groups consist of indistinguishable pairs obtained in previous step and individual distinguishable states. The groups obtained are the states of minimized DFA.

Step 3: The minimized DFA can be obtained as shown below. If $[p_1, p_2,\ldots p_k]$ is one group, then $\delta([p_1,p_2,\ldots p_k], a) = [r_1, r_2,\ldots r_m]$ is also a member of the group obtained earlier and place an edge from the group $[p_1,p_2,\ldots p_k]$ to the group $[r_1, r_2,\ldots r_m]$ and label the edge with the symbol *a*. Follow this procedure for each group obtained in step 2 and for each $a \in \Sigma$.

Step 4: The start state of the minimized DFA is the group $[p_1,p_2,...p_k]$ provided this group contains the start state of given DFA. If the group $[p_1,p_2,...p_k]$contains a final state of given DFA, then the group $[p_1,p_2,...p_k]$ is final state of minimized DFA.

Example 4.15: Minimize the following DFA.

	Σ	
States	0	1
→A	B	F
B	G	C
$\textcircled{C}$	A	C
D	C	G
E	H	F
F	C	G
G	G	E
H	G	C

Step 1: Obtain the distinguishable and indistinguishable states using the *table-filling* algorithm (mark procedure). The table is shown below: (see the example 4.14).

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

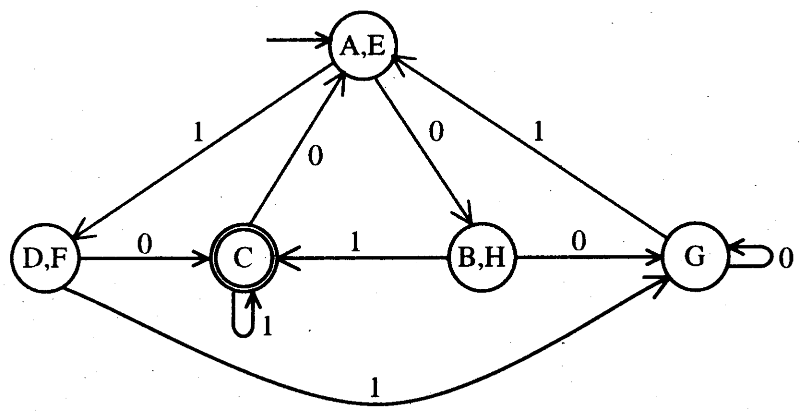
Step2: The unmarked pairs (A,E) (B,H) and (D,F) are indistinguishable and the states C and G are distinguishable. So, the various groups that represent the states of minimized DFA are:

(A,E), (B,H), C, (D,F), G

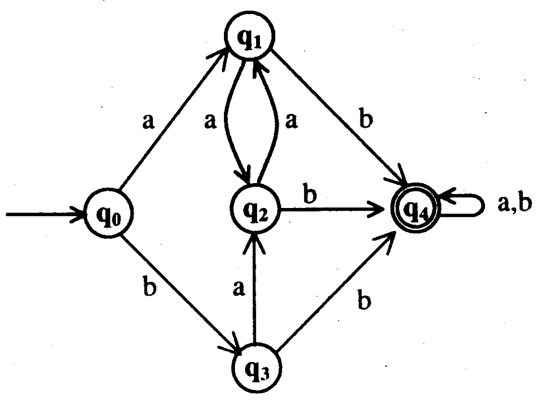
Step 3: The minimized transition table is shown below:

	Σ	
States	0	1
→ (A,E)	(B,H)	(D,F)
(B,H)	G	C
Ⓒ	(A,E)	C
(D,F)	C	G
G	G	(A,E)

Note that the start state of minimized DFA is the group which has the start state of DFA to be minimized and the group which consists of a final state of original DFA is the final state of minimized DFA. The transition diagram of the minimized DFA is shown below:



Example 4.16: Minimize the following DFA.



Solution: First let us obtain the table for inequalities for the sates (distinguishable or indistinguishable states) . This we can do by knowing the various states of given DFA. The states of the DFA are:

$q_0 \quad q_1 \quad q_2 \quad q_3 \quad q_4$

The pairs can be obtained as shown below:(This can be obtained by writing all the pairs in the matrix form and take only the upper triangular matrix to avoid duplicate elements and cycles)

$(q_0, q_1)\,(q_0, q_2)\,(q_0, q_3)\,(q_0, q_4)$
$(q_1, q_2)\,(q_1, q_3)\,(q_1, q_4)$
$(q_2, q_3)\,(q_2, q_4)$
(q_3, q_4)

The above pairs can be represented in the tabular form as shown below and note that no pairs are marked.

q_1				
q_2				
q_3				
q_4				
	q_0	q_1	q_2	q_3

Step 1:

For each pair (p, q) where p ∈ Q and q ∈ Q, if p ∈ F and q ∉ F or vice versa then, the pair (p, q) is distinguishable and mark the pair (p, q) [by putting say 'x' for the pair (p,q)]. This clearly indicates that one of the pair should be a final state and the other should be a non final state. So, the various pairs which satisfy the above criteria are all the pairs involving the final state q_4 and are marked as shown below:

q_1				
q_2				
q_3				
q_4	x	x	x	x
	q_0	q_1	q_2	q_3

Step 2:

For each pair (p, q) and for each $a \in \Sigma$ find $\delta(p, a) = r$ and $\delta(q, a) = s$. If the pair (r, s) is already marked as distinguishable (Note: The distinguished states are already marked as 'x' in the first step) then the pair (p, q) is also distinguishable and mark it as say 'x'. Repeat step 2 until no previously unmarked pairs are marked

Now consider the pairs (p, q) which are not marked in step 1 and obtain the corresponding pairs (r, s) on input symbols 0 and 1 as shown below:

δ	**a**	**b**
(p,q)	**(r,s)**	**(r,s)**
(q_0,q_1)	(q_1,q_2)	**(q_3,q_4)**
(q_0,q_2)	(q_1,q_1)	**(q_3,q_4)**
(q_0,q_3)	(q_1,q_2)	**(q_3,q_4)**
(q_1,q_2)	(q_2,q_1)	(q_4,q_4)
(q_1,q_3)	(q_2,q_2)	(q_4,q_4)
(q_2,q_3)	(q_1,q_2)	(q_4,q_4)

Note: If the pairs (r, s) on *a* and *b* are marked in step 1, the corresponding pair (p, q) should be marked. For example, for the pair (q_0,q_1) on b is (q_3,q_4) and is marked as 'x' in step 1. So, mark the pair (q_0,q_1) (shown using shading). On similar lines identify the pairs (r, s) which are marked in step 1 (shown in bold phase) and take the corresponding pairs (p,q) (shown in shading) and update the marking table as shown below.

q_1	x			
q_2	x			
q_3	x			
q_4	x	x	x	x
	q_0	q_1	q_2	q_3

Again consider the pairs (p,q) which are not marked in the above table and see whether the corresponding pairs (r, s) on a and b are marked as shown below:

δ	**a**	**b**
(p,q)	**(r,s)**	**(r,s)**
(q_1,q_2)	(q_2,q_1)	(q_4,q_4)
(q_1,q_3)	(q_2,q_2)	(q_4,q_4)
(q_2,q_3)	(q_1,q_2)	(q_4,q_4)

None of the unmarked pairs (r, s) are marked and final table is shown below:

q_1	x			
q_2	x			
q_3	x			
q_4	x	x	x	x
	q_0	q_1	q_2	q_3

Observe that the pairs (q_1,q_2), $(q_1,q_3)$and $(q_2,q_3)$are not at all marked and are considered to be indistinguishable states. Note that the state q_1 is present in the pairs (q_1,q_2), (q_1,q_3) and the state q_2 is present in the pairs (q_1,q_2), (q_2,q_3). So, the states q_1, q_2, q_3 together forms a group consisting of indistinguishable states and can be represented as (q_1, q_2, q_3). So, the indistinguishable groups are

(q_1, q_2, q_3)

and distinguishable states are

(q_0) and (q_4)

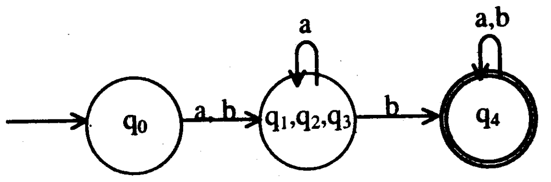
Step2: So, the various groups that represent the states of minimized DFA are:

$(q_0)\ (q_1, q_2, q_3)\ (q_4)$

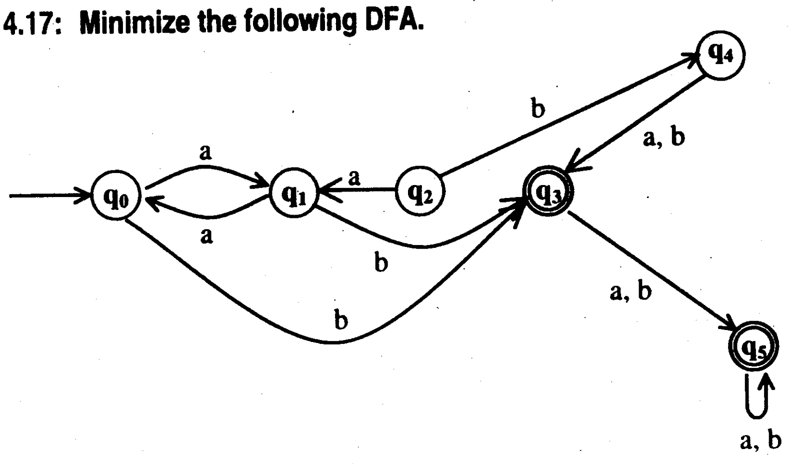
Step 3: The minimized transition table is shown below. Note that the start state of original DFA is the start state of minimized DFA and the group which consists of a final state of original DFA is the final state of minimized DFA.

	Σ	
States	a	b
$\rightarrow (q_0)$	(q_1,q_2,q_3)	(q_1,q_2,q_3)
(q_1,q_2,q_3)	(q_1,q_2,q_3)	(q_4)
(q_4)	(q_4)	(q_4)

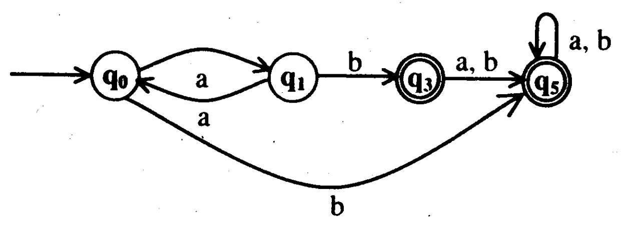
The transition diagram obtained after minimization is shown below:



Example 4.17: Minimize the following DFA.



Note: It is clear from the above that the states q_2 and q_4 are not reachable from the start state and they can be removed. The resulting DFA after removing those states is shown below:



First let us obtain the table for inequalities for the sates (distinguishable or indistinguishable states) . This we can do by knowing the various states of given DFA. The states of the DFA are:

$q_0 \quad q_1 \quad q_3 \quad q_5$

The pairs can be obtained as shown below:(This can be obtained by writing all the pairs in the matrix fórm and take only the upper triangular matrix to avoid duplicate elements and cycles)

$(q_0, q_1)\,(q_0, q_3)\,(q_0, q_5)$
$(q_1, q_3)\,(q_1, q_5)$
(q_3, q_5)

The above pairs can be represented in the tabular form as shown below and note that no pairs are marked.

q_1			
q_3			
q_5			
	q_0	q_1	q_3

Step 1:

For each pair (p, q) where $p \in Q$ and $q \in Q$, if $p \in F$ and $q \notin F$ or vice versa then, the pair (p, q) is distinguishable and mark the pair (p, q) [by putting say 'x' for the pair (p,q)]. This clearly indicates that one of the pair should be a final state and the other should be a non final state. So, the various pairs which satisfy the above criteria are all the pairs involving the one of the final states q_3 or q_5 i.e., (q_0, q_3), (q_0, q_5), (q_1, q_3) and (q_1, q_5) and are marked as shown below:

q_1			
q_3	x	x	
q_5	x	x	
	q_0	q_1	q_3

Step 2:

For each pair (p, q) and for each $a \in \Sigma$ find $\delta(p, a) = r$ and $\delta(q, a) = s$. If the pair (r, s) is already marked as distinguishable (Note: The distinguished states are already marked as 'x' in the first step) then the pair (p, q) is also distinguishable and mark it as say 'x'. Repeat step 2 until no previously unmarked pairs are marked

Now consider the pairs (p, q) which are not marked in step 1 and obtain the corresponding pairs (r, s) on input symbols a and b as shown below:

δ	**a**	**b**
(p,q)	**(r,s)**	**(r,s)**
(q_0,q_1)	(q_1,q_0)	(q_3,q_3)
(q_3,q_5)	(q_5,q_5)	(q_5,q_5)

Note: If the pairs (r, s) on *a* and *b* are marked in step 1, the corresponding pair (p, q) should be marked. But, the pairs (r, s) on *a* and *b* are not marked in the previous step and the final table that provides the inequalities between the various states are shown below:

q_1			
q_3	X	X	
q_5	X	X	
	q_0	q_1	q_3

Observe that the pairs (q_0,q_1) and (q_3,q_5) are not at all marked and are considered to be indistinguishable states.

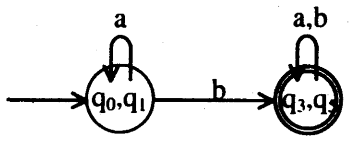
Step 3: Finding the states of minimized DFA. It is clear from the table obtained in step 2 that the pairs (q_0,q_1) and (q_3,q_5) are indistinguishable and the various groups that represent the states of minimized DFA are:

(q_0, q_1) (q_3, q_5)

Step 4: The minimized transition table is shown below. Note that the start state of minimized DFA is the group which has the start state of DFA to be minimized and the group which consists of a final state of original DFA is the final state of minimized DFA.

	Σ	
States	a	b
$\rightarrow(q_0,q_1)$	(q_0,q_1)	(q_3,q_5)
$*(q_3,q_5)$	(q_3,q_5)	(q_3,q_5)

The transition diagram obtained after minimization is shown below:



4.12 Alternate procedure to minimize the states of DFA

Step 1:

Remove all unreachable states from the start state.

Step 2:

Construction of Π_{old}: Partition the states into two groups one consisting of final states and the other consisting of non-final states.

Step 3:

Construct Π_{new} from Π_{old} as shown in the figure 4.3.

Step 4:

```
If $\Pi_{new}$ != $\Pi_{old}$
        Copy $\Pi_{new}$ to $\Pi_{old}$
        goto step 3
endif
```

Step 5:

Pick a representative from each group Π (Π can be Π_{new} or Π_{old} as they are equal). These representatives will be the states of reduced DFA. Obtain the transitions for the reduced DFA as shown below:

Let (p, q) (r, s)....(t, u) be the groups in Π. In each group, we can select any state as the representative. Suppose, *p*, *s* and *u* are the representatives. If $\delta(p, a) = r$ is the transition then we can write $\delta(p, a) = s$. This is because, even though *r* is the transition, since it is not a representative we use its representative i.e., s. Using this procedure construct the transition table or transition graph.

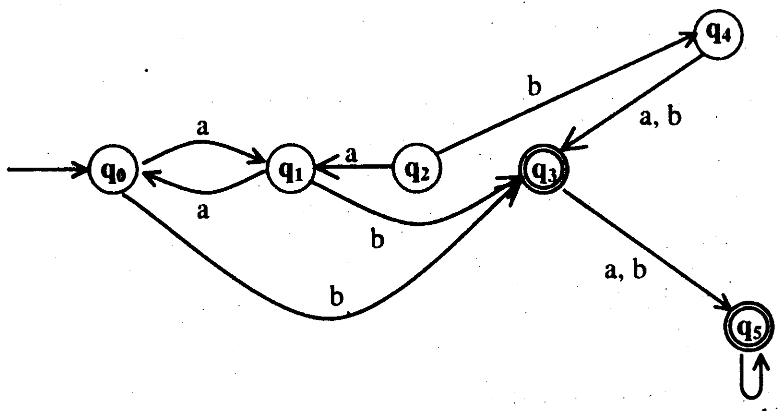
Procedure to construct Π_{new} from Π_{old}.

```
Copy $\Pi_{old}$ to $\Pi$

for each $a$ in $\Sigma$ do
```

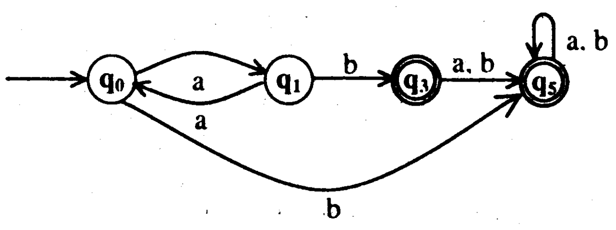
```
    for each subgroup G in Π
        Find δ(q, a) for each q in G on input symbol a.

        Suppose $q_i$, $q_j$, $q_k$, $q_l$, $q_m$ and $q_n$ are the states in subgroup G and $p_i$,
        $p_j$, $p_k$, $p_l$, $p_m$ and $p_n$ are the resulting states on an input symbol a.

        If $p_i$, $p_j$, $p_k$, $p_l$, $p_m$ and $p_n$ all belongs to one specific subgroup in
        $\Pi_{old}$, then there is no need of partition.

        If $p_i$,$p_l$ belongs to one group, $p_j$,$p_k$ belongs to another group and $p_m$,
        $p_n$ belongs to some other group, then partition the subgroup G into
        groups consisting of ($q_i$, $q_l$), ($q_j$, $q_k$) and ($q_m$, $q_n$).
    end for

    Let the resulting subgroups be Π.

end for

Copy the final Π to $\Pi_{new}$.
```

Fig. 4.3 Procedure to obtain Π_{new} from Π_{old}

Example 4.18: Minimize the following DFA.



Note: It is clear from the above that the states q_2 and q_4 are not reachable from the start state and they can be removed. The resulting DFA after removing those states is shown below:



Step 2:

The states of the DFA shown in figure above can be partitioned into two groups: one consisting of non final states and the other group consisting of only final states as shown below:

$\Pi_{old} = (q_0, q_1)\ (q_3, q_5) = (G_1, G_2)$ ---A

Step 3:

Obtain Π_{new} from Π_{old} as shown below:

On input *a*:

Consider the group: (q_0, q_1):

The transitions from each state on input symbol *a* are:

$$\left.\begin{array}{l}\delta(q_0, a) = q_1 \\ \delta(q_1, a) = q_0\end{array}\right\}$$ All belong to group G_1 and so there is no need to partition the group G_1

Consider the group: (q_3, q_5):

The transitions from each state on input symbol *a* are:

$$\left.\begin{array}{l}\delta(q_3, a) = q_5 \\ \delta(q_5, a) = q_5\end{array}\right\}$$ Both transitions enters into state q_5 and so there is no need to partition the group G_2.

So, $\Pi = (q_0, q_1)\ (q_3, q_5)$

On input *b*:

Consider the group: (q_0, q_1):

The transitions from each state on input symbol *b* are:

$$\left.\begin{array}{l}\delta(q_0, b) = q_3 \\ \delta(q_1, b) = q_3\end{array}\right\}$$ Both transitions enters into state q_3 and so there is no need to partition the group G_1.

<u>Consider the group: (q_3, q_5):</u>

The transitions from each state on input symbol *b* are:

$$\left.\begin{array}{l}\delta(q_3, b) = q_5\\ \delta(q_5, b) = q_5\end{array}\right\}$$ Both transitions enters into state q_5 and so there is no need to partition the group G_2.

So, $\Pi = (q_0, q_1)\ (q_3, q_5)$

After copying Π to Π_{new} we have, $\Pi_{new} = (q_0, q_1)\ (q_3, q_5)$------------------------------B

By comparing (A) and (B), it is very clear that Π_{new} is equal to Π_{old}. Since they are same, the representatives from each group will be the states of reduced DFA. The representatives are q_0 from the first group and q_3 from the second group. Therefore, the states of reduced DFA are q_0 and q_3 and the transitions for these states can be obtained as shown below:

Consider state q_0:

$\delta(q_0, a) = q_1$. But, q_1 belongs to group G_1 and the representative of this group is q_0. So, instead of q_1 we can write q_0. Therefore,

$\delta(q_0, a) = q_0$ (C)

$\delta(q_0, b) = q_3$ (D)

Consider state q_3:

$\delta(q_3, a) = q_5$ which is equal to q_3 since q_3 and q_5 are in the same group and q_3 is the representative of that group. So,

$\delta(q_3, a) = q_3$ (E)

$\delta(q_3, b) = q_5$ which is equal to q_3. Note that q_3 and q_5 are in the same group and q_3 is the representative of that group. So,

$\delta(q_3, b) = q_3$ (F)

The transition table for the transitions obtained from (C) to (F) are shown in the table 4.3 and the reduced DFA using the transition graph is shown in figure 4.4. Here, q_3 is the final state since it is the final state in the given DFA.

	δ	←Σ→ a	b
States	$\rightarrow q_0$	q_0	q_3
	q_3	q_3	q_3

Table 4.3 Transition table

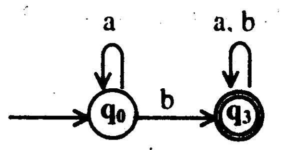
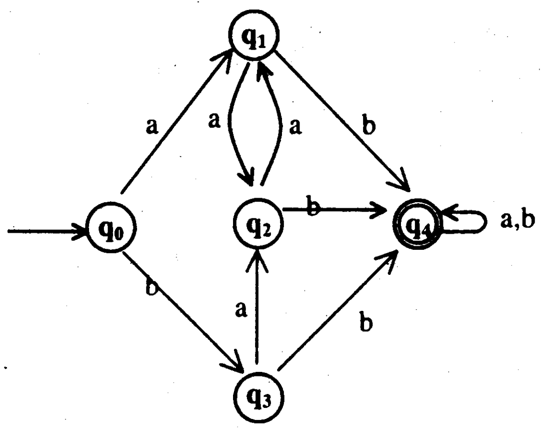


Fig. 4.4 Reduced DFA

Example 4.19: Minimize the following DFA.



Step 1:

Since all the states reachable from q_0, there is no need to remove any state.

Step 2:

The states can be partitioned into two groups: one consisting of non final states and the other group consisting of only final states as shown below:

$$\Pi_{old} = (q_0, q_1, q_2, q_3)\ (q_4) = (G_1, G_2) \qquad \text{---A}$$

Step 3:

Since G_2 contain only one state, the group G_2 need not be considered.

Obtain Π_{new} from Π_{old} as shown below:

On input *a*:

Consider the group: (q_0, q_1, q_2, q_3):

The transitions from each state on input symbol *a* are:

$$\left.\begin{array}{l}\delta(q_0, a) = q_1\\ \delta(q_1, a) = q_2\\ \delta(q_2, a) = q_1\\ \delta(q_3, a) = q_2\end{array}\right\}$$ All the states obtained belong to group G_1 and so there is no need to partition the group G_1

So, $\Pi = (q_0, q_1, q_2, q_3)\ (q_4)$

On input *b*:

Consider the group: (q_0, q_1, q_2, q_3):

The transitions from each state on input symbol *b* are:

$$\left.\begin{array}{l}\delta(q_0, b) = q_3\\ \delta(q_1, b) = q_4\\ \delta(q_2, b) = q_4\\ \delta(q_3, b) = q_4\end{array}\right\}$$ From the states q_1, q_2 and q_3 there is a transition to state $q_4 \in G_2$ and there is a transition from q_0 to $q_3 \in G_1$

So, partition the group G1 into two subgroups
$(q_0)\ (q_1, q_2, q_3)$

So, $\Pi = (q_0)\ (q_1, q_2, q_3)\ (q_4)$

After copying Π to Π_{new} we have,
$\Pi_{new} = (q_0)\ (q_1, q_2, q_3)\ (q_4)$ --- B

Compare A and B i.e., Compare Π_{old} and Π_{new}. Since they are not same copy Π_{new} to Π_{old} and repeat step 3 again.

Step 3:

$\Pi_{old} = (q_0)\ (q_1, q_2, q_3)\ (q_4) = (G_1, G_2, G_3)$ --- C
Obtain Π_{new} from Π_{old} as shown below:

Since the group G_1 and G_3 contain single state, those groups need not be considered.

On input *a*:

Consider the group: (q_1, q_2, q_3):
The transitions from each state on input symbol *a* are:

$$\left.\begin{array}{l}\delta(q_1, a) = q_2\\ \delta(q_2, a) = q_1\\ \delta(q_3, a) = q_2\end{array}\right\}$$ All the states obtained belong to group G_2 and so there is no need to partition the group G_2

So, $\Pi = (q_0)\,(q_1, q_2, q_3)\,(q_4)$

On input *b*:

Consider the group: (q_1, q_2, q_3):

The transitions from each state on input symbol *b* are:

$$\left.\begin{array}{l}\delta(q_1, b) = q_4\\ \delta(q_2, b) = q_4\\ \delta(q_3, b) = q_4\end{array}\right\}$$ From the states q_1, q_2 and q_3 there is a transition to state $q_4 \in G_2$ and so there is no need to partition G_2.

So, $\Pi = (q_0)\,(q_1, q_2, q_3)\,(q_4)$

After copying Π to Π_{new}, we have,

$\Pi_{new} = (q_0)\,(q_1, q_2, q_3)\,(q_4)$ --- D

Compare C and D i.e., Compare Π_{old} and Π_{new}. Since $\Pi_{new} = \Pi_{old}$ goto step 5.

Step 5:

$\Pi_{old} = \Pi_{new} = \Pi = (q_0)\,(q_1, q_2, q_3)\,(q_4) = (G_1, G_2, G_3)$

Pick a representative from each group. The representatives are q_0 from the first group, q_1 from the second group and q_4 from the third group. These representatives are the states of reduced DFA. Therefore, the states of reduced DFA are

q_0, q_1 and q_4

and the transitions for these states can be obtained as shown below: