Chapter 4

Regular Expression and Languages

Inside This Chapter

4.1. Regular Expression

We have already studied Finite Automata. Now we switch our attention from machine like description of language to an algebraic description : "the regular expression." We shall find that, regular expressions can define exactly the same languages that the various forms of automata describe : the regular languages.

However, regular expression offer something that automata do not : a declarative way to express the strings, we want to accept. Thus regular expression as the input language for many system. For example, lexical-analyser generators, such as Lex or Flex.

4.1.1. The Operators of Regular Expression

Before describing the regular notation, we need to learn the three operations on languages that operators of regular expression represent. These operations are :

1. The union of two languages L_1 and L_2, denoted $L_1 \cup L_2$, is the set of string that are in either L_1 or L_2, or both.

   For example, if $L_1 = \{001, 10, 111\}$ and $L_2 = \{\epsilon, 001\}$ then $L_1 \cup L_2 = \{\epsilon, 10, 001, 111\}$.

2. The concatenation of languages L_1 and L_2 is a set of strings that can be formed by taking any string in L_1 and concatenating it with any strings in L_2. We denote concatenation of languages with a dot.

   For example, if $L_1 = \{001, 10, 111\}$ and $L_2 = \{\epsilon, 001\}$, then $L_1 \cdot L_2 = \{001, 10, 111, 001001, 10001, 111001\}$.

3. The closure (or star, or kleene closure) of a language L is denoted by L^* and represents the set of those strings that can be formed by taking any number of strings from L, possibly with repetitions (that is same string can be repeated more then once) and concatenating all of them. For example, $L = \{0, 1\}$, then L^* is all strings of 0's and 1's including null string.

   More formally, L^* is the infinite union $U_i \geq L^i$, where $i \geq 0$.

4. The positive closure of a language L is denoted by L^+ and represent the set of those strings that can be formed by taking any number of strings from L, possibly with repetitions and concatenating all of them, excluding null string *i.e.*

$$L^+ = L^* - \epsilon$$

More formally, L^+ is the infinite union $U_i \geq 1\ L^i$, where $i > 0$.

4.1.2. Definition of Regular Expression

Definition The set of regular expression is defined by the following rules :

1. Every letter of Σ can be made into a regular expression, null string, ϵ itself is a regular expression.
2. If r_1 and r_2 are regular expression, then
   (*i*) (r_1) (*ii*) $r_1 r_2$
   (*iii*) $r_1 + r_2$ (*iv*) r_1^*
   (*v*) r_1^+ are also regular expression.
3. Nothing else is regular expression.

4.1.3. Building Regular Expression

The algebra of regular expressions follows this path using constants and variables that denote languages, and operators for three operations of union, dot and star. We can describe the regular expression recursively, as follows :

In this definition, we not only describe what are the legal regular expression, but for each regular expression r, we describe the language, it represents, which we denote as $L(r)$. We can consider some facts :

1. The constants ϵ (null string) and ϕ (empty set) are regular expression, denoting the languages $\{\epsilon\}$ and ϕ, respectively.

   That is, $L(\epsilon) = \{\epsilon\}$, and $L(\phi) = \phi$.

2. If a is any symbol, then a is regular expression. This expression denotes the language $\{a\}$. That is $L(a) = \{a\}$.

3. A variable, usually capitalized and italic such as L is a variable, representing any language.

4.1.4. Inductive Step

There are four parts to the inductive step, one for each of three operators and one for the introduction of parentheses.

1. If r_1 and r_2 are regular expressions, then $r_1 + r_2$ is a regular expression denoting the union of $L(r_1)$ and $L(r_2)$.

That is, $$L(r_1 + r_2) = L(r_1) \cup L(r_2)$$

2. If r_1 and r_2 are regular expression, then $r_1 r_2$ is a regular expression denoting the concatenating of $L(r_1)$ and $L(r_2)$

That is, $$L(r_1 r_2) = L(r_1) \,.\, L(r_2)$$

3. If r is a regular expression then r^* is a regular expression denoting the closure of $L(r)$. That is,

$$L(r^*) = (L(r))^*$$

4. If r is a regular expression, then (r), a parenthesized r, is also a regular expression, denoting the same language as r.

Formally, $$L((r)) = L(r).$$

Now let us discuss some examples of building regular expression for better understanding.

Example 4.1. Write the regular expression over alphabet (a, b, c) containing at least one a and at least one b.

Solution. Let us first analyse the language of regular expression.

Problem is very simple and the language will have set of strings, like *ab*, *abbbb*, *abab*, *aaaab*, *baaaa* ... . So strings are 1 *a* 2 *b* 3, any combination of *a*'s, *b*'s and *c*'s is possible at the places 1, 2, and 3 any combination of *a*, *b*, *c* means $(a + b + c)^*$.

So regular expression is

$$r = (a + b + c)^* a(a + b + c)^* b(a + b + c)^* + (a + b + c)^* b(a + b + c)^* a(a + b + c)^*$$

If $r = r_1 + r_2$ (say)

then minimum string of r_1 is *ab* and minimum string of r_2 is *ba*.

Example 4.2. Write the regular expression for the set of strings of 0's and 1's whose tenth symbol from the right end is 1.

Solution. $\Sigma = \{0, 1\}$ given the set of strings, according to the required regular expression are 1110000000000, 1001101010101010 .... . So clearly we concern about only tenth position and it should be 1, rest 9 places from the right are

either 0 or 1. Similarly positions 11th onward from right end may be either 0 or 1.

So regular expression is

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

$$r = (0 + 1)^* 1(0 + 1)^9.$$

Example 4.3. Write the regular expression for the set of strings of 0's and 1's such that in every prefix, the number of 0's equal number of 0's and 1's such that in every prefix, the number of 0's differs from the number of 1's by at most 1.

Solution. Clearly the set of strings is having strings like 10101010, 10101010, 0110, 1001 ...

So regular expresion will be

$$r = (01 + 10)^*.$$

Example 4.4. Write a regular expression for the set of strings of 0's and 1's not containing 101 as a substring.

Solution. When we analyse the set of strings then it becomes clear that string may start with '0' and 0's may be repeated, wherever one is encountered then a single 0 must follow it so strings are like 0000010, 00000111111, 1001001 ....

Regular expression is $r = (0^*1^*00)^* 0^* 1^*.$

Example 4.5. Write the regular expression over alphabet $\{a, b\}$ for the set of strings with even number of a's followed by odd number of b's that is for the language

$$L = \{a^{2n}b^{2m+1} : n \geq 0, m \geq 0\}.$$

Solution. Clearly every string of language L will start with even number of a's. String should end with odd number of b's.

Regular expression is $r = (aa)^* (bb)^* b.$

Example 4.6. Write the regular expression for the language

$L = \{w \in \{0, 1\}^* : w$ has no pair of consecutive zeros$\}$.

Solution. Clearly whenever a 0 occurs, it must be followed by a 1. Such a substring may be proceeded and followed by any number of 1's, that is $(1^*011^*)^*$. But strings ending in 0 or consisting of all 1's are unaccounted here, that is $1^* (0 + \epsilon)$.

So regular expression is $r_1 = (1^*011^*)^* (0 + \epsilon) + 1^*(0 + \epsilon)$

Another possible solution is $r_2 = (1 + 01)^* (0 + \epsilon)$

"Although the two expression look different, both answers are correct, as they denote the same language. Generally, there are an unlimited number of regular expressions for any given language."

Example 4.7. Write the regular expression for the language

$$L = \{a^n b^m / n \geq 4, m \leq 3\}$$

Solution. Language contain the set of strings such that every string start with atleast 4 a's and end at the most 3 b's.

So regular expression will be $r = a^4\ a^*\ (\in + b + b^2 + b^3)$.

Example 4.8. Write the regular expression for the language

$$L = \{a^n\ b^m : (n + m) \text{ is even}\}$$

Solution. We know $(n + m)$ can be even in two cases

Case 1. n and m both are even.

Case 2. n and m both are odd.

Let regular expression for case 1 is r_1 then

$$r_1 = (aa)^*\ (bb)^*$$

Let regular expression for case 2 is r_2 then

$$r_2 = (aa)^*\ a\ (bb)^*\ b$$

Let regular expression for language L is r

then

$$r = r_1 + r_2$$

$$r = (aa)^*\ (bb)^* + (aa)^*\ a\ (bb)^*\ b.$$

Example 4.9. Write the regular expression for the language

$$L = \{ab^n\ w : n \geq 3,\ w \in (a, b)^+\}$$

Solution. Clearly every string of language L starts with a followed by atleast three b's, followed by atleast one a or one b, that is strings are like ab^3a, $abbbbbba$, $abbbbbbbb$, $abbbb....a$

So regular expression is : $r = ab^3b^*\ (a + b)^+$

Here + is a positive closure *i.e.* $(a + b)^+ = (a + b)^* - \in$.

Example 4.10. Write the regular expression for the language

$$L = \{w : |\ w\ |\ \text{mod } 3 = 0\},\ w \in (a, b)^*\}$$

Solution. Let us first understand the meaning of $|\ w\ |\ \text{mod } 3 = 0$, when length of string belongs to the language L, is divided by 3 then reminder should be 0, that is length of w must be 0, 3, 6, 9 ... . Clearly multiple of three.

Regular expression is : $r = ((a + b)^3)^*$

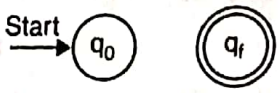
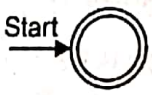
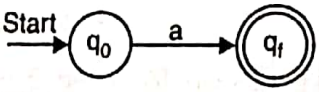
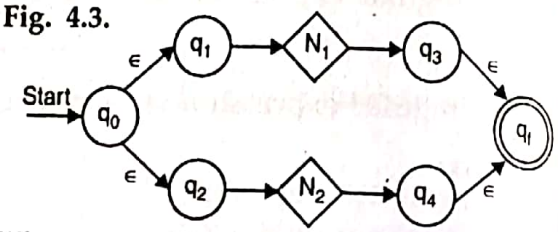
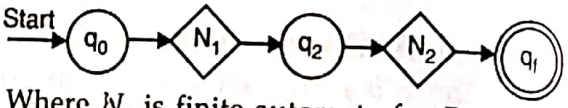
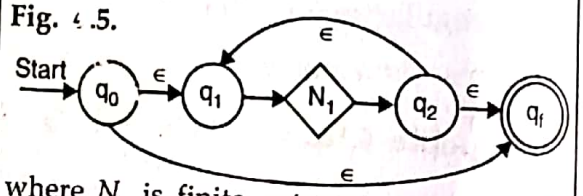
Example 4.11. Write the regular expression for the language

$$L = \{w \in (a, b)^* : n_a(w) \text{ mod } 3 = 0\}$$

Solution. $n_a(w)$ mod 3 = 0 means, number of a's in string should be 0, 3, 6, 9 ... .

So regular expression is : $r = (b^*ab^*ab^*ab^*)^*$

4.2. Comparative Study of Regular Expression, Regular Sets and Finite Automata

Regular expression	*Regular set*	*Finite automata*
ϕ	$\{\ \}$	 **Fig 4.1. q_f : is the final state.**
$\in$	$\{\in\}$	 **Fig. 4.2.**
Every a in Σ is a regular expression.	$\{a\}$	 **Fig. 4.3.**
If r_1 and r_2 are regular expression then $(r_1 + r_2)$ or r_1/r_2 is regular expression.	$R_1 \cup R_2$ (where R_1 and R_2 are regular sets corresponding to r_1 and r_2 respectively).	 Where N_1 is FA accepting R_1 and N_2 is a finite automata accepting R_2. **Fig. 4.4.**
r_1r_2 is a regular expression.	R_1R_2 (where R_1 and R_2 are regular sets corresponding to r_1 and r_2 respectively).	 Where N_1 is finite automata for R_1 and N_2 is FA for R_2. **Fig. 4.5.**
r^* is a regular expression if r is regular expression.	R^* (where R is a regular set corresponding to r).	 where N_1 is finite automata accepting R. **Fig. 4.6.**

4.3. Construction of FA for Regular Expression

We can construct NFA with $\in$ transmission by considering following four facts.

1. The expression is $r + s$ for some smaller expression r and s. The following automation serves, where R is automation for r and S is automation for s. That is, starting at the new start state, we can go to the start state of either the automation for r or the automation for S. We can reach the accepting state of one of these automata, following a path labelled by some string in $L(r)$ and $L(s)$, respectively. Once we reach the accepting state of the automation for r or s, we can follow one of the $\in$-arcs to the accepting state of the new automation. The language for the automation in Fig. 4.7 (a) is $L(r) \cup L(s)$.

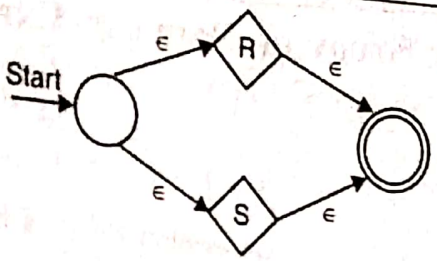


Fig. 4.7 (*a*).

2. The expression *rs* for some smaller expression *r* and *s*. The automation for the concatenation is show in Fig. 4.7 (*b*).

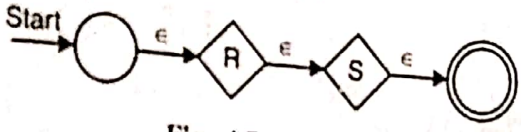


Fig. 4.7 (*b*).

TIPS

Note that the start state of the first automation becomes the start state of the whole, and the accepting state of the second automation becomes the accepting of the whole. The idea is that the only paths from start to accepting state to go first through the automation of *r*, where it must follow a path labelled by a string in *L*(*r*). and then through the automation for *s*, where it follows a path labelled by a string in *L*(*s*). Thus, the path in the automation of Fig. 4.7 (*b*) are all and only those labelled by strings in *L*(*R*) *L*(*s*).

3. The expression is r^* for some smaller expression *r*. Then we use automation of Fig. 4.7 (*c*). That automation allow us to go either :

(*i*) Directly from the start state to the accepting state along a path labelled ∈.

(*ii*) To the start state of automation for *r*, through that automation one or more times, and then to accepting state. This set of paths allow us to accept strings in *L*(*r*), *L*(*r*), *L*(*r*), *L*(*r*), *L*(*r*), *L*(*r*), and so on, thus covering all strings in $L(r^*)$ except perhaps ∈ which was covered by the direct arc to the accepting state.

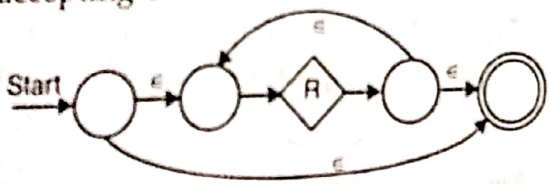


Fig. 4.7 (*c*).

where *R* is automation for *r*.

4. The expression is (*r*) for the smaller expression *r*. The automation for *r* also serves as the automation for (*r*), since the parentheses do not change the language defined by the expression.

Example 4.12. One the basis of above discussion find the automation for regular expression $a \cdot (a + b)^* \cdot b \cdot b$.

Solution. Let us solve this problem step by step :

Step 1. The basic regular expression involved are *a* and *b*, we start with automation for *a* and automation for *b*. Since brackets are evaluated first. We first constructed automation for $(a + b)$, using automation for *a* and automation is shown in Fig. 4.8(*a*)

Fig. 4.8(*a*). Automation for $(a + b)$.

Step 2. Since closure is required to take next, we construct automation for $(a + b)^*$ using automation for $(a + b)$ as shown in Fig. 4.8(*b*).

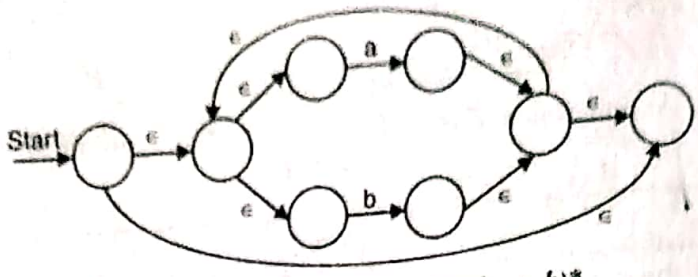


Fig. 4.8(*b*). Automation for $(a + b)^*$

Step 3. Next we construct the automation for $a \cdot (a + b)^*$ as shown in Fig. 4.8 (c)

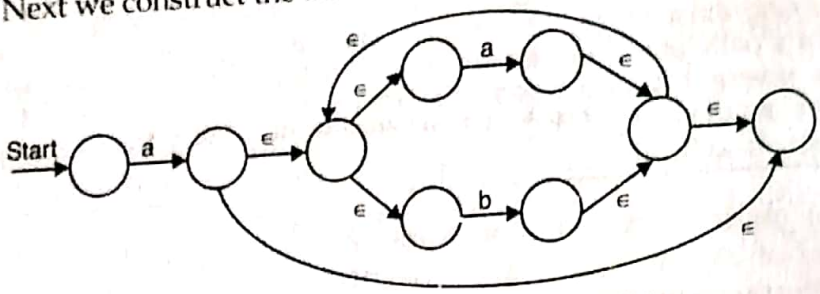


Fig. 4.8(*c*). Automation for $a \cdot (a + b)^*$.

Step 4. Next we construct the automation for $a \cdot (a + b)^* \cdot b$ by using automation for $a \cdot (a + b)^*$.

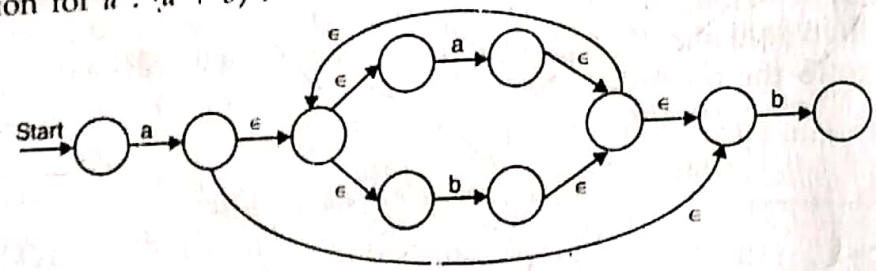


Fig. 4.8(*d*). Automation for $a \cdot (a + b)^* \cdot b$.

Step 5. Now finally we can construct automation for $a \cdot (a + b)^* \cdot b \cdot b$ by using automation for $a \cdot (a + b)^* \cdot b$.

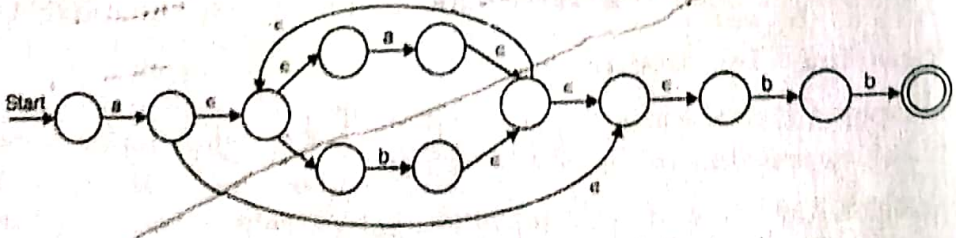


Fig. 4.8(*e*). Automation for $a \cdot (a + b)^* \cdot b \cdot b$.

It is required automation for given regular expression.

Example 4.13. Construct the ∈-NFA for the regular expression $(0 + 1)^* 1 (0 + 1)$.

Solution. Here automation for 0 and 1 is basic automation. First we will make automation for $(0 + 1)$ as in Fig. 4.9 (*a*).

Now we will convert it into automation for $(0 + 1)^*$ as shown in Fig. 4.9(*b*).

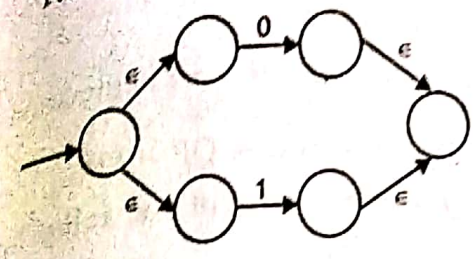


Fig. 4.9(*a*). Automation for (0 + 1).

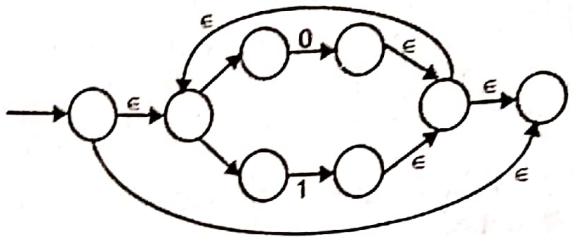


Fig. 4.9(*b*). Automation for $(0 + 1)^*$.

Now we will construct automation for 1 . (0 + 1) as shown in Fig. 4.9(*c*).

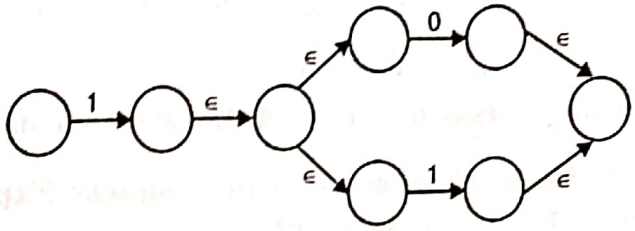


Fig. 4.9 (*c*). Automation for 1 . (0 + 1).

Now we have to connect ∈-NFA of Fig. 4.9(*b*) and ∈-NFA of Fig. 4.9 by a ∈-transition to find out the ∈-NFA for given regular expression as shown in Fig. 4.9(*d*).

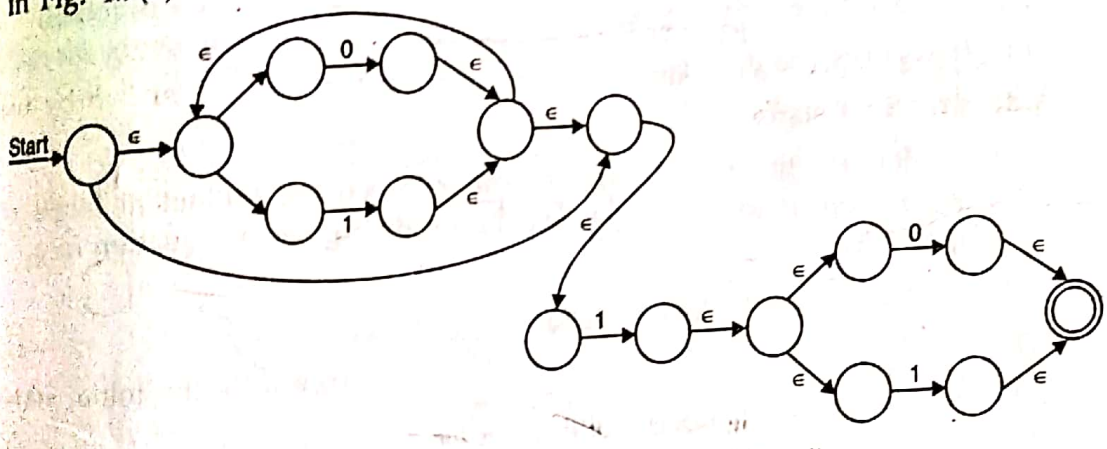


Fig. 4.9 (*d*). Automation for $(0 + 1)^* 1(0 + 1)$.

4.4. Construction of Regular Expression from DFA

Now it is clear that regular expression is mathematical expression for a given regular language. We have already know that for every regular language there exist a deterministic finite automate. We can conclude that regular expression, regular language for that regular expression and deterministic finite automata for that regular language are similar things in different representation. So we can construct regular expression for every deterministic automata.

4.4.1. Ardens's Theorem

Let P and Q be two regular expression over alphabet Σ. If P does not contain null string $\in$, then

$$R = Q + RP$$

has a unique solution that is $R = QP^*$

It can be understand as : $R = Q + RP$

Put the value of R in R.H.S.

$$R = Q + (Q + RP)P = Q + QP + RP^2$$

When we put the value of R again and again we got the following equation.

$$R = Q + QP + QP^2 + QP^3 \ldots$$
$$R = Q(1 + P + P^2 + P^3 \ldots$$
$$R = Q(\in + P + P^2 + P^3 + \ldots$$
$$R = QP^*$$

(By the definition of closure operation for regular expression.)

4.4.2. Use of Arden's Theorem to find Regular Expression of a Deterministic Finite Automata

There are certain assumptions which are made regarding the transition system :

(*i*) The transition diagram should not have $\in$-transitions.

(*ii*) It must have only a single initial state.

(*iii*) It vertices are $q_1 \ldots q_n$.

(*iv*) q_i is final state.

(*v*) w_{ij} denotes the regular expression representing the set of labels of edges from q_i to q_j. We can get the following set of equation in $q_1 \ldots q_n$.

$$q_1 = q_1w_{11} + q_2w_{21} + \ldots + q_nw_{n1} + \in \quad \ldots(1)$$

(since q_1 is the initial state)

$$q_2 = q_1w_{12} + q_2w_{22} + \ldots q_nw_{n2} \quad \ldots(2)$$
$$\vdots \qquad\qquad \vdots$$
$$q_n = q_1w_{1n} + q_2w_{2n} + \ldots + q_nw_{nn} \quad \ldots(n)$$

We solve these equation for q_i in terms of w_{ij}'s and it will be required regular expression. One thing should be noted that we add $\in$ (null string) in the equation starts with starting state q_1 and we solve equation to find out q_i (final state) in terms of w_{ij}'s, it is one of the regular expression for given deterministic finite automata.

Example 4.14. Find the regular expression for transition diagram given in Fig. 4.11.

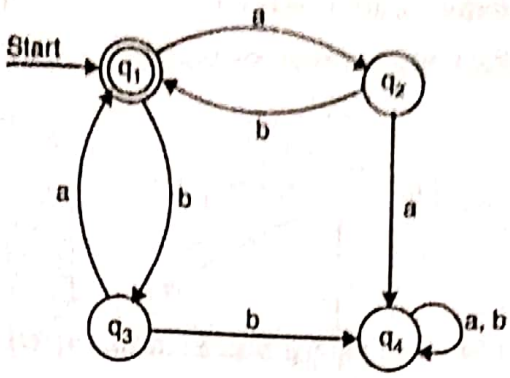


Fig. 4.11.

Solution. Now let us from the equations :

$$q_1 = q_2b + q_3a + \in$$
$$q_2 = q_1a$$
$$q_3 = q_1b$$
$$q_4 = q_2a + q_3b + q_4a + q_4b$$

Put q_2 and q_3 in q_1 as

$$q_1 = q_1ab + q_1ba + \in$$
$$q_1 = \in + q_1(ab + ba)$$
$$q_1 = \in(ab + ba)^*$$

So required regular expression is $(ab + ba)^*$.

Example 4.15. Construct a regular expression corresponding to the state diagram given in Fig. 4.12.

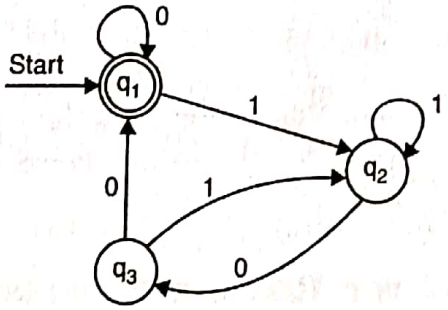


Fig. 4.12.

Solution. Let us form the equations :

$$q_1 = q_10 + q_30 + \in$$
$$q_2 = q_11 + q_21 + q_31$$
$$q_3 = q_20$$
$$q_2 = q_11 + q_21 + (q_20)\ 1 = q_11 + q_2(1 + 01)$$
$$q_2 = q_1\ 1\ (1 + 01)^*$$

So
$$q_1 = q_10 + q_30 + \in = q_10 + q_200 + \in = q_10 + (q_11\ (1+01)^*)00 + \in$$
$$q_1 = q_1(0 + 1(1 + 01)^*00) + \in$$
$$q_1 = \in(0 + 1(1 + 01)^*00)^*$$
$$q_1 = (0 + 1(1 + 01)^*\ 00)^*$$

So regular expression is $(0 + 1(1 + 01)^* 00)^*$.

Example 4.16. Find the regular expression corresponding to Fig. 4.13.

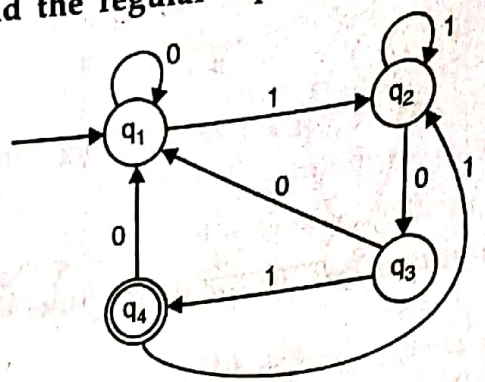


Fig. 4.13.

Solution. Now let us from the equations

$$q_1 = q_10 + q_30 + q_40 + \epsilon \quad (1)$$
$$q_2 = q_11 + q_21 + q_41 \quad (2)$$
$$q_3 = q_20 \quad (3)$$
$$q_4 = q_31 \quad (4)$$

(4) $\Rightarrow$ Now $q_4 = q_31 = q_201$

(2) $\Rightarrow$ Using q_2 equation, we get $q_2 = q_11 + q_21 + q_2011 = q_11 + q_2(1 + 011)$

$$q_2 = q_11(1 + 011)^* = q_1(1(1 + 011)^*)$$

Put q_3 and q_4 in q_1 equation $q_1 = q_10 + q_200 + q_2010 + \epsilon$

$$= q_10 + q_2(00 + 010) + \epsilon$$
$$= q_10 + q_11(1 + 011)^* (00 + 010) + \epsilon$$
$$q_1 = \epsilon(0 + 1(1 + 011)^* (00 + 010))^*$$

Put q_2 in q_4 equation $q_4 = q_11(1 + 011)^* 01 = q_1(1(1 + 011)^*01)$

$$q_4 = (0 + 1(1 + 011)^* (00 + 010))^* (1(1 + 011)^*01)$$

So required regular expression is

$$(0 + 1(1 + 011)^* (00 + 010))^* (1(1 + 011)^*01)$$

4.5. Algebraic Laws for Regular Expressions

In this section, we will see a collection of algebraic laws that bring the issue of when two regular laws are equivalent. Instead of examining specific regular expressions, we shall consider pairs of regular expressions with variables as arguments.

TIPS

Two expressions with variables are equivalent if whatever languages we substitute for the variables, the result of the two expressions are the same language.

Like arithmetic expressions, the regular expressions have a number of laws that work for them. Many of these are similar to the laws for arithmetic, if we think of union as addition and concatenation as multiplication. However, there are a few places where the analogy breaks down, and there are also some laws that apply to regular expressions but have no analog for arithmetic.

especially when the closure operator is involved. The next sections from a catalogue of the major laws.

4.5.1. Associativity and Commutativity

Commutativity is the property of an operator that says that we can switch the order of its operands and get the same result. For example, the associative law of multiplication is $(x \times y) \times z = x \times (y \times z)$. Here are three laws of these types that hold for regular expression :

$$P + Q = Q + P.$$

This law, the commutative law for the union, says that we take union of two languages in either order.

$$(P + Q) + R = P + (Q + R)$$

This law, the associative law for union says that we may take the union of three languages either by taking the union of the first two initially, or taking the union of the last two initially.

Note that, together with the commutative law for union, we conclude that we can take the union of any collection of languages with any order and grouping, and the result will be the same. Intuitively, a string is in $L_1 \cup L_2 \cup \ldots + L_K$ if and only if it is in one or more of the L_i's.

$$(PQ)R = P(QR)$$

This law, the associative law for concatenation, says that we can concatenate three languages by concatenating either for first two or the last two initially.

4.5.2. Identities

An identity for an operator is a value such that when the operator is applied to identity and some other value, the result is the same value. For instance, 0 is the identity for addition, since $0 + m = m + 0 = m$, and 1 is the identity for the multiplication, since $1 \times x = x \times 1 = x$. There are three laws for regular expression involving these concepts; we list them below.

- $\phi + L = L + \phi = L$. This law assets that ϕ is the identity for union.
- $\epsilon L = L\epsilon = L$. This law assets that ϵ is the identity for concatenation.
- $\phi L = L\phi = \phi$. This law assets that ϕ is the annihilater for concatenation.

4.5.3. Distributed Law

A distributed law involves two operators, and assets that one operator can be pushed down to be applied to each argument of the operator individually. The most common example is the distributed law of multiplication over addition, that is

$$x \times (y + z) = x \times y + x \times z$$

However, there is an analogous law for regular expression, that we most state in two forms. Since concatenation is not commutative. These laws are :

(i) $P(Q + R) = PQ + PR$. This law, is the left distributed law of concatenation over union.

(ii) $(Q + R)P = QP + RP$. This law, is the right distributed law of concatenation over union.

4.5.4. The Idempotent Law

An operator is said to be idempotent if the result of applying it to two of the same values as arguments is that value. The common arithmetic operators are not idempotent, $x + x \neq x$ in general and $x \times x \neq x$ in general (although there are some values of x for which the equality holds, such as $0 + 0 = 0$). However, union and intersection are common examples of idempotent operators. Thus for the regular expression, we may assert the following law :

- $L + L = L$. This law, the idempotent law for union, states if we take the union of two identical expression, we can replace them by one copy of the expression.

4.5.5. Laws for Closure

There are number of laws involving the closure operators. We shall list them here, and give some explanation for why they are true.

1. $(L^*)^* = L^*$. This law says that closing an expression that is already closed does not change the language. The language of $(L^*)^*$ is all strings created for concatenating strings in the language L^*. But those strings are themselves composed of strings from L. Thus, the string in $(L^*)^*$ is also a concatenation of strings from L and therefore in the language of L^*.
2. $\phi^* = \epsilon$. The closure of ϕ contains only the string ϵ.
3. $\epsilon^* = \epsilon$. It is easy to check that only string that can be formed by concatenating any number of copies of the empty string is the empty string itself.
4. $L^+ = LL^* = L^*L$. Recall that positive closure of L, L^+ is defined to be $L + LL + LLL + \ldots$. Also, $L^* = \epsilon + L + LL + LLL + \ldots$

   Thus $LL^* = L + LL + LLL + LLLL + \ldots$

   That proves $L^+ = LL^*$.
5. $L^* = L^+ + \epsilon$. The proof is easy, since the expansion of L^+ includes every term in the expansion of L^* except ϵ. Note that if the language L contains the string ϵ then the additional $+\epsilon$ is not needed; that is $L^+ = L^*$ in this special case.
6. $L? = \epsilon + L$. This rule is really the definition of the operator ?

4.5.6. Identities for Regular Expression

On the basis of above discussion we will see some identitites for the regular expression. Let p, q, r are three regular expression. Two regular expression p and q are equivalent (we say $p = q$) if p and q represent the same set of strings.

Some identities are following :

I^1 $\quad \phi + r = r$

I^2 $\quad \phi r = r\phi = \phi$

I^3 $\quad \epsilon r = r\epsilon = r$

I^4 $\quad \epsilon^* = \epsilon$ and $\phi^* = \epsilon$

I^5 $\quad r + r = r$

I^6 $\quad r^*r^* = r^*$

I^7 $\quad rr^* = r^*r = r^+$

I^8 $\quad (r^*)^* = r^*$

I^9 $\quad \in + rr^* = r^* = \in + r^*r$

I^{10} $\quad (pq)^*p = p(qp)^*$

I^{11} $\quad (p + q)^* = (p^*q^*)^* = (p^* + q^*)^*$

I^{12} $\quad (p + q)r = pr + qr$ and $r(p + q) = rp + rq$

Example 4.17. Prove that $(1 + 00^*1) + (1 + 00^*1)(0 + 10^*1)^*(0 + 10^*1) = 0^*1(0 + 10^*1)^*$

Solution.

L.H.S. $= (1 + 00^*1)(\in + (0 + 10^*1^*(0 + 10^*1)$ using I^{12}

$= (1 + 00^*1)(0 + 10^*1)^*$ using I^9

$= (\in + 00^*)1(0 + 10^*1)^*$ using I^{12} for $(1 + 00^*1)$

$= 0^*1(0 + 10^*1)^*$ using I^9

$=$ R.H.S.

Hence it is proved.

Example 4.18. Prove $t + RR^* = R^*$ where t is null string and R is a regular expression.

Solution. Two regular expressions P and Q are equivalent (we write $P = Q$). If P and Q are represent the same set of strings.

From Arden's Theorem, we know that, the solution of equation $R' = Q + R'P$ is $R' = QP^*$

Now $\quad t + RR^* = R^*$

L.H.S. $= t + RR^*$ Compare it with Arden's theorem

$\simeq Q + R'P$

$Q = t$

$R' = R$

$P = R^*$

So, the solution of this equation will be $R' = QP^*$

$R_1 = QP^*$

$= tR^*$

$= R^*$

So we can write that $t + RR^* = R^*$

Example 4.19. Prove the following identity : $(a^*ab + ba)^* a^* = (a + ab + ba)^*$.

Solution.

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

By taking L.H.S. $= (a * ab + ba)^* a^*$

$= (a\ (ab)^* + (ba)^*)^*a^*$

$= (ab^* + (ba)^*\ a^*)$

$= (ab^* + (ba)^*\ a^*)$

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

$=$ R.H.S.