1/12/2023

MODULE - 2 (8 hours)

Regular Expressions and Regular Languages.

Refer 3.1 to 3.3, Chapter 4

3.1* Regular Expressions
3.1.1→ The operators of Regular Expressions
3.1.2→ Building Regular expressions.
3.2.1→ From DFA's to Regular Expressions
3.2.2→ Converting DFA's to Regular Expressions by Eliminating states
3.2.3→ Converting Regular Expressions to Automata.
* Regular Languages.
→ Regular languages
4.1.1→ Proving languages not to be regular languages.
4.4→ Equivalence and minimization of automata.

3.1 Regular Expressions

Now, we switch our attention from machine-like descriptions of languages - deterministic and non-deterministic finite automata - to an algebraic description: the "Regular expression".

Regular expressions serve as the input language for many systems that process strings.

Examples include:

1. Search commands such as the UNIX grep or equivalent commands for finding strings that one sees in web browsers or text-formatting systems. These systems use a regular-expression-like notation for describing patterns that user wants to find in a file. Different search systems convert the regular expression into either a DFA or an NFA, and simulate that automaton on the file being searched.

2. Lexical-analyzer generators, such as Lex or Flex. (lexical analyzer is the component of a compiler that breaks the source program into logical units (called tokens) of one or more characters that have a shared significance.

Examples of tokens include keywords (eg. while), identifiers (eg. any letter followed by zero or more letters and/or digits), and sign as + or <=.

A lexical-analyzer generator accepts descriptions of the forms of tokens, which are essentially regular expressions, and produces a DFA that recognizes which token appears next on the input.

NOTE:-

1. The language accepted by finite automata can easily described by simple expression called Regular expression.
2. It is most effective way to represent any language.
3. The language accepted by some regular expression are referred to as Regular languages.
4. A regular expression can also be described as a sequence of pattern that defines a string.
5. For instance:

Ex:- if R.E x^* means zero or more occurance of x.
It can generate $\{\epsilon, x, xx, xxx, \ldots\}$

Ex:- if R.E x^+ means one or more occurence of x.
It can generate $\{x, xx, xxx, \ldots\}$.

3.1.1. The operators of Regular expressions

Regular expressions denote languages.
For example, the R.E: $01^* + 10^*$ denotes the language consisting of all strings that are either a single 0 followed by any number of 1's or a single 1 followed by any number of 0's.

The three operations on languages that the operators of regular expressions represent are explained below:-

1. The Union of two languages L and M, denoted L U M, the set of strings that are in either L or M, or both.

for example, if $L = \{001, 10, 111\}$ and $M = \{\epsilon, 001\}$, then $L \cup M = \{\epsilon, 10, 001, 111\}$.

2. The concatenation of Languages L and M is the set of strings that can be formed by taking any string in L and concatenating it with any string in M.
i.e, define the concatenation of a pair of strings → one string is followed by the other to form the result of the concatenation.
Concatenation operator is frequently called "dot".
For example, if $L = \{001, 10, 111\}$ and $M = \{\epsilon, 001\}$, then L.M or just LM, is $= \{001, 10, 111, 001001, 10001, 111001\}$.
The First three strings in LM are the strings in L Concatenated with ϵ. Since ϵ is the identity for Concatenation, the resulting strings are the same as the strings of L. However, the last three strings in LM are formed by taking each string in L and concatenating it with the second string in M, which is 001.

3. The Closure (or Star, or Kleene closure) of a Language L is denoted L^* and represents the set of those strings that can be formed by taking any number of strings from L, possibly with repetitions (i.e, the same string may be selected more than once) and concatenating all of them.

For instance, if $L = \{0, 1\}$, then L^* is all strings of 0's and 1's.

If $L = \{0, 11\}$, then L^* consists of those strings of 0's and 1's such that the 1's come in pairs, eg, 011, 11110, and ϵ, but not 01011 or 101.

More formally, L^* is the infinite union $\bigcup_{i \geq 0} L^i$, where $L^0 = \{\epsilon\}$, $L^1 = L$, and L^i, for $i > 1$ is $LL \cdots L$ (the concatenation of i copies of L).

Example 1:- Let $L = \{0, 11\}$

$L^0 = \{\epsilon\}$, independent of what Language L is; the 0^{th} power represents the selection of zero strings from L. $L^1 = L$, which represents the choice of one string from L. Thus, the first two terms in the expansion of L^* give us $\{\epsilon, 0, 11\}$.

$\{111\}$ Consider L^2. pick two strings from L, with repetitions allowed, so there are four choices.

$L^2 = \{00, 011, 110, 1111\}$.

$\{1111\}$ L^3 is the set of strings that may be formed by making three choices of the two strings in L and gives

$L^3 = \{000, 0011, 0110, 1100, 01111, 11011, 11110, 111111\}$.

To compute L^*, we must compute L^i for each i, and take union of all these languages. L^i has 2^i members.

$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$ is L in the particular case that the language L is the set of all strings of 0's.

Example 2 : $\emptyset^* = \{\epsilon\}$. Note that $\emptyset^0 = \{\epsilon\}$, while $\emptyset^i$ for any $i \geq 1$, is empty, since we can't select any strings from the empty set.

In fact, $\emptyset$ is one of only two languages whose closure is not infinite.

3.1.2 Building Regular Expressions

Algebras of all kinds start with some elementary expressions, usually constants and/or variables.

Algebras then allow us to construct more expressions by applying a certain set of operators to these elementary expressions and to previously constructed expressions.

Usually, some method of grouping operators with their operands, such as parentheses, is required as well.

For instance, Arithmetic algebra starts with constants such as integers and real numbers, plus variables, and builds more complex expressions with arithmetic operators such as + and ×.

The algebra of regular expressions follows this pattern, using constants and variables that denote languages, and operators.

Describe the regular expressions recursively, as follows. In this definition, we not only describe what the legal R.E are, but for each R.E E, we describe the language it represents, which we denote L(E).

BASIS: The basis consists of three parts:

1. The constants ϵ and ϕ are R.E, denoting the languages $\{\epsilon\}$ and ϕ, respectively.
   i.e, $L(\epsilon) = \{\epsilon\}$, and $L(\phi) = \phi$.
2. If a is any symbol, then a is R.E. This expression denotes the language $\{a\}$. i.e, $L(a) = \{a\}$.
3. A variable, usually capitalized and italic such as L, is a variable, representing any language.

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

1. If E and F are R.E, then E+F is R.E denoting the union of L(E) and L(F).
   i.e, $L(E+F) = L(E) \cup L(F)$.
2. If E and F are R.E, then EF is a R.E denoting concatenation of L(E) and L(F).
   i.e, $L(EF) = L(E)L(F)$.

For instance, 0.1 is R.E meaning the same as 01 and representing the language $\{01\}$.
However avoid the dot as concatenation in R.E.

3. If E is a R.E, then E^* is a R.E, denoting the closure of L(E). i.e, $L(E^*) = (L(E))^*$.
4. If E is a R.E, then (E), a parenthesized E, is also a R.E, denoting the same language as E.
   Formally, $L((E)) = L(E)$.

Example 1 :-
Consider the language consisting of strings of a's and b's containing aab.
Soln :- R.E is $(a+b)^* aab (a+b)^*$.

3.1.3 Precedence of Regular-Expression operators.

Like other algebras, the R.E operators have an

assumed order of "precedence", which means that operators are associated with their operands in a particular order.

For instance, W.K.T $xy + z$ groups the product xy before the sum.
i.e, $(xy) + z$. and not to the expression $x(y+z)$.

Illy, we group two of the same operators from the left in arithmetic, so $x - y - z$ is equivalent to $(x-y) - z$, and not to $x - (y - z)$.

For R.E, the following is the order of precedence for the operators:

1. The star operator is of highest precedence.
i.e, it applies only to the smallest sequence of symbols to its left that is a well-formed R.E.

2. Next in precedence comes the concatenation or "dot" operator.
Concatenation is an associative operator.

3. Finally, all unions (+ operators) are grouped with their operands. Since is also associative.

Example 1 :- The expression $01^* + 1$ is grouped $(0(1^*)) + 1$.

Soln:- The star operator is grouped first. Since the symbol 1 immediately to its left is a legal regular expression, that alone is the operand of the star.
Next, group the concatenation between 0 and (1^*), giving the expression $(0(1^*))$.
Finally, the union operator connects the latter expression and the expression to its right, which is 1.

Problems :-

(1.) Write R.E for the following languages over $\{0,1\}^*$.

a) The set of all strings that begin with 110.

Soln :-

b) The set of all strings that contain 1011.

Soln;-

c) The set of all strings that contain exactly three 1's.

Soln :-

(2). Write R.E for the language accepting all strings containing any number of a's and b's.

Soln :- R.E = $(a+b)^*$.

$L = \{\epsilon, a, b, aa, bb, aba, bab, \ldots\}$

This will given the set as above.

the $(a+b)^*$ shows any concatenation with a and b. even a null string.

(3) Write R.E for the language accepting all the string which are starting with 1 and ending with 0, over $\Sigma = \{0,1\}$.

Soln :- The first symbol is 1 and last symbol is 0.

$\therefore$ R.E = $1(0+1)^*0$.

(4) Write R.E for the language starting with a but not having consecutive b's.

Soln;- R.E = $\{a+ab\}^*$

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

(5) Write R.E for the following languages.

a) The set of all strings that the number of 0's is odd.

Soln:-

b) The set of all strings that do not contain 1101.

Soln:-

(6) Write R.E for the following languages.

a) The set of all strings of 0's and 1's not containing 101 as a substring.

Soln:-

b) The set of all strings with an equal number of 0's and 1's, such that no prefix has two more 0's than 1's, nor two more 1's than 0's.

Soln:-

c) The set of strings of 0's and 1's whose number of 0's is divisible by five and whose number of 1's is even.

Soln:-

(7) Give english descriptions of the languages of the following regular expressions:

a) $(1+\epsilon)(00^*1)^*0^*$

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

c) $(0+10)^*1^*$.

(8.) Write the R.E for the Language over $\Sigma = \{0\}$ having even length of the string.

Soln:- $L = \{ \epsilon, 00, 0000, 00000, \ldots \}$

$R.E = (00)^*$

(9.) Write R.E for the language L over $\Sigma = \{0,1\}$ such that all the string do not contain the substring 01.

Soln:- $L = \{ \epsilon, 01, 00, 11, 10, 100, \ldots \}$

$R.E = (1^*0^*)$

3.2 Finite Automata and Regular Expressions.

While the R.E approach to describing languages is fundamentally different from the finite-automaton approach, these two notations turn out to represent exactly the same set of languages, which we have termed the "Regular languages".

We have already shown that DFA, and the two kinds of NFA - with and without ϵ-transitions - accept the same class of languages.

In order to show that the R.E define the same class, we must show that:

1. Every language defined by one of these automata is also defined by a R.E. For this proof, we can assume the language is accepted by some DFA.
2. Every language defined by a R.E is defined by one of these automata. For this part of the proof, the easiest is to show that there is an NFA with ϵ-transitions accepting the same language.

Figure. shows all equivalences we have proved or will prove. An arc from class X to class Y means that we prove every language defined by class X

is also defined by class Y. Since the graph is strongly connected (i.e, we can get from each of the four nodes to any other node).

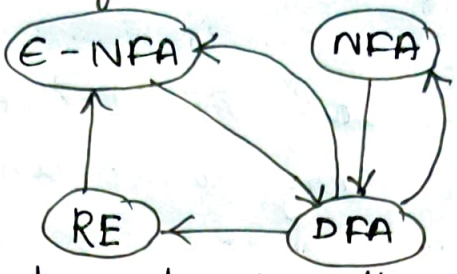


Figure. Plan for showing the equivalence of four different notations for regular languages.

3.2.1 From DFA's to Regular Expressions:

The construction of a R.E to define the language of any DFA is suprisingly tricky. Roughly, we build expressions that describe sets of strings that label certain paths in the DFA's transition diagram.

However, the paths are allowed to pass through only a limited subset of the states.

In an inductive definition of these expressions, start with the simplest expression that describe paths that are not allowed to pass through any states (i.e, they are single nodes or single arcs), and indictively build the expressions that let the paths go through progressively larger sets of states.

Finally, the paths are allowed to go through any state; i.e, the expressions we generate at the end represent all possible paths. These ideas appear in the proof of the following theorem.

Theorem 3.4: If $L = L(A)$ for some DFA A, then there is a R.E R such that $L = L(R)$.

Proof:- Let us suppose that A's states are $\{1, 2, \ldots n\}$ for some integer n.

There will be n of them for some finite n, and by renaming the states, we can refer to the states in this manner, as if they were the first n positive integers.

Construct a collection of R.E that describe progressively broader sets of paths in the transition diagram of A.

Let us use $R_{ij}^{(k)}$ as the name of a R.E whose language is the set of strings w such that w is the label of a path from state i to state j in A, and that path has no intermediate node whose number is greater than k.

Beginning and end points of the path are not "intermediate", so there is no constraint that i and/or j be less than or equal to k.

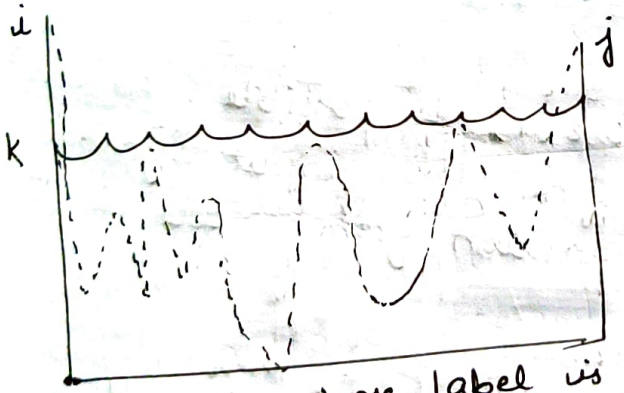


Figure. A path whose label is in the language of R.E $R_{ij}^{(k)}$.

Figure suggests the requirement on the paths represented by $R_{ij}^{(k)}$. There, the vertical dimension represents the state, from 1 at the bottom to n at the top, and the horizontal dimension represents travel along the path.

In the figure, we have shown both i and j to be greater than k, but either or both could be less or

k

Also, the path passes through node k twice, but never goes through a state higher than k, except at the end points.

To construct the expressions $R_{ij}^{(k)}$, the following inductive definition, starting at k=0 and finally reaching k=n.

Notice that, when k=n, there is no restriction at all on the paths represented, since there are no states greater than n.

BASIS: The basis is k=0. Since all states are numbered 1 or above, the restriction on paths is that the path must have no intermediate states at all.

There are only two kinds of paths that meet such a condition:

1. An arc from node (state) i to node j.
2. A path of length 0 that consists of only some node i.

If $i \neq j$, then only case (1) is possible. We must examine the DFA A and find those input symbols a such that there is a transition from state i to state j on symbol a.

a) If there is no such symbol a, then $R_{ij}^{(0)} = \phi$.

b) If there is exactly one such symbol a, then $R_{ij}^{(0)} = a$.

c) If there are symbols $a_1, a_2, \ldots, a_k$ that label arcs from state i to state j, then

$$R_{ij}^{(0)} = a_1 + a_2 + \cdots + a_k.$$

If $i = j$, then expression becomes $\epsilon + a_1 + a_2 + \ldots + a_k$ (i.e, the path of length=0 is represented by R.E as ϵ).

INDUCTION :- Suppose there is a path from state i to state j that goes through no state higher than k.

There are two possible cases to consider:

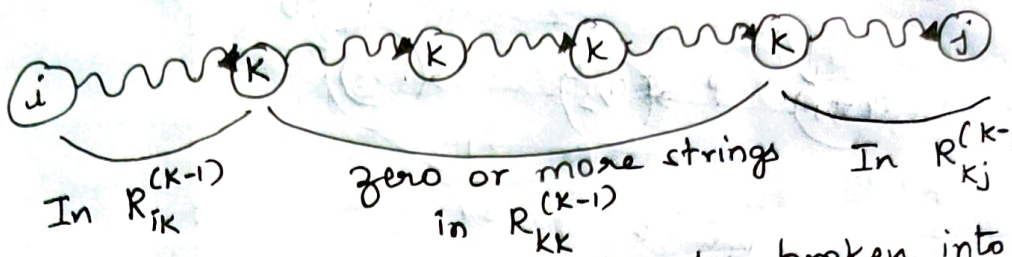


Figure: A path from i to j can be broken into segments at each point where it goes through state K.

1. The path does not go through state K at all. In this case, the label of the path is in the language of $R_{ij}^{(k-1)}$.

2. The path goes through state K at least once. Then we can break the path into several pieces, as shown in above figure.

Note that if the path goes through state K only once, then there are no "middle" pieces, just a path from i to K and a path from K to j.

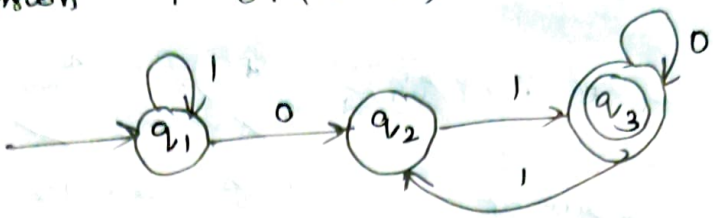
The set of labels for all paths of this type is represented by the R.E $R_{ik}^{(k-1)} \left(R_{kk}^{(k-1)}\right)^* R_{kj}^{(k-1)}$.

- $R_{ik}^{(k-1)}$: Expression Represents the part of path that gets to state K the first time.
- $\left(R_{kk}^{(k-1)}\right)^*$: portion that goes from K to itself, zero times, once or more than once.
- $R_{kj}^{(k-1)}$: the part of the path that leaves K for the last time and goes to state j.

$$\therefore \boxed{R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} \left(R_{kk}^{(k-1)}\right)^* R_{kj}^{(k-1)}}$$

Eventually, we have $R_{ij}^{(n)}$ for all i and j. We may assume that state 1 is the start state, although the accepting states could be any set of the states. The R.E for the language of the automaton is then the sum (union) of all expressions $R_{ij}^{(n)}$ such that state j is an accepting state.

Example 1 :- Convert the DFA to R.E. Clearly this DFA accepts all strings corresponding to the regular expression $1^*01(0+11)^*$.



Soln :-

FORMULAS

NOTE :- Basis : $k=0$.

(1) $i \neq j$

(i) $\textcircled{i}$ $\textcircled{j}$ $\quad R_{ij}^{(0)} = \phi$

(ii) $\textcircled{i} \xrightarrow{a} \textcircled{j}$ $\quad R_{ij}^{(0)} = a$.

(iii) $\textcircled{i} \xrightarrow{a_1, a_2, \ldots, a_k} \textcircled{j}$ $\quad R_{ij}^{(0)} = a_1 + a_2 + \ldots + a_k$.

(2) $i = j$

(i) $\textcircled{i}$ $\quad R_{ij}^{(0)} = \epsilon$

(ii) $\textcircled{i}$ with self-loop a $\quad R_{ij}^{(0)} = \epsilon + a$.

(iii) $\textcircled{i}$ with self-loop $a_1, a_2, \ldots a_k$ $\quad R_{ij}^{(0)} = \epsilon + a_1 + a_2 + \ldots + a_k$.

$R_{11}^{(0)}$	$1+\epsilon$
$R_{12}^{(0)}$	0
$R_{13}^{(0)}$	ϕ
$R_{21}^{(0)}$	ϕ
$R_{22}^{(0)}$	ϵ
$R_{23}^{(0)}$	1
$R_{31}^{(0)}$	ϕ
$R_{32}^{(0)}$	1
$R_{33}^{(0)}$	$0+\epsilon$

(a)

$R_{11}^{(1)}$	1^*
$R_{12}^{(1)}$	1^*0
$R_{13}^{(1)}$	ϕ
$R_{21}^{(1)}$	ϕ
$R_{22}^{(1)}$	ϵ
$R_{23}^{(1)}$	1
$R_{31}^{(1)}$	ϕ
$R_{32}^{(1)}$	1
$R_{33}^{(1)}$	$0+\epsilon$

(b)

$R_{11}^{(2)}$	1^*
$R_{12}^{(2)}$	1^*0
$R_{13}^{(2)}$	1^*01
$R_{21}^{(2)}$	ϕ
$R_{22}^{(2)}$	ϵ
$R_{23}^{(2)}$	1
$R_{31}^{(2)}$	ϕ
$R_{32}^{(2)}$	1
$R_{33}^{(2)}$	$0+\epsilon+11$

(c)

Table/Figure. R.E for paths that can go through (a) no state (b) state 1 only, (c) states 1 and 2 only.

The above table and figure give various stages of the construction. The basis expression are shown in the first table.

from Inductive Rule,

$$R_{ij}^{(1)} = R_{ij}^{(0)} + R_{i1}^{(0)} (R_{11}^{(0)})^* R_{1j}^{(0)} \longrightarrow (1)$$

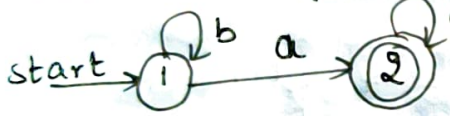
$$R_{ij}^{(2)} = R_{ij}^{(1)} + R_{i2}^{(1)} (R_{22}^{(1)})^* R_{2j}^{(1)} \longrightarrow (2)$$

$$R_{13}^{(3)} = R_{13}^{(2)} + R_{13}^{(2)} (R_{33}^{(2)})^* R_{33}^{(2)} \longrightarrow (3)$$ where $i = 1$, $j = 3$ and $K = 2$.

$$= 1^* 01 + 1^* 01 (0 + \epsilon + 11)^* (0 + \epsilon + 11)$$

$$= 1^* 01 (0 + 11)^*$$

(2). Conversion of DFA to R.E. (By Rijk method)



Soln:- Step1:- $R_{12}^{(2)}$ ← no. of states.

↑ start state ↑ final state

(i.e, the Regular Expression R contain start state 1 and final state 2 and k means total no. of states in DFA i.e $k = 2$.)

Step2:- Initially, Start from the Basis condition. i.e, $K = 0$. Consider the 6 Basis formulas given in the note.

$K = 0$.

$R_{11}^{(0)}$	$\epsilon + b$
$R_{12}^{(0)}$	a
$R_{21}^{(0)}$	ϕ
$R_{22}^{(0)}$	$\epsilon + a + b$

← $i = j$ ∴ $R_{ij}^{(0)}$ = ϵ + transition ⇒ $R_{11}^{(0)} = \epsilon + b$.

← $i \neq j$ ∴ $R_{ij}^{(0)}$ = transition ⇒ $R_{12}^{(0)} = a$.

Step3:- for $K = 1$, refer Inductive formulae.

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

$$R_{ij}^{(1)} = R_{ij}^{(0)} + R_{i1}^{(0)} (R_{11}^{(0)})^* R_{1j}^{(0)}.$$

now construct table by considering $k=1$ and also the previous table (i.e $k=0$) for refelence.

By direct substitution apply the formulae.

$$R_{ij}^{(1)} = R_{ij}^{(0)} + R_{i1}^{(0)} \left(R_{11}^{(0)}\right)^* R_{1j}^{(0)}$$

$K=1$

		By direct substitution	Simplification
$i=1$ $j=1$	$R_{11}^{(1)}$	$(\epsilon+b) + (\epsilon+b)(\epsilon+b)^* (\epsilon+b)$	$(\epsilon+b) + b^* = b^*$
$i=1$ $j=2$	$R_{12}^{(1)}$	$a + (\epsilon+b)(\epsilon+b)^* a$	$a + b^* a = b^* a$
$i=2$ $j=1$	$R_{21}^{(1)}$	$\phi + \phi(\epsilon+b)^* (\epsilon+b)$	ϕ
$i=2$ $j=2$	$R_{22}^{(1)}$	$(\epsilon+a+b) + \phi(\epsilon+b)^* a$	$(\epsilon+a+b)$

Step 4 :- Now, For $k=2$

$$R_{ij}^{(2)} = R_{ij}^{(1)} + R_{i2}^{(1)} \left(R_{22}^{(1)}\right)^* R_{2j}^{(1)}.$$

		By direct substitution	Simplification
$i=1$ $j=1$	$R_{11}^{(2)}$	$b^* + b^* a (\epsilon+a+b)^* \phi$	b^*
$i=1$ $j=2$	$R_{12}^{(2)}$	$b^* a + b^* a (\epsilon+a+b)^* (\epsilon+a+b)$	$b^* a + b^* a (\epsilon+a+b)^*$ $= b^* a (a+b)^*$
$i=2$ $j=1$	$R_{21}^{(2)}$	$\phi + (\epsilon+a+b)(\epsilon+a+b)^* \phi$	ϕ
$i=2$ $j=2$	$R_{22}^{(2)}$	$(\epsilon+a+b) + (\epsilon+a+b)(\epsilon+a+b)^*$ $(\epsilon+a+b)$	$(\epsilon+a+b) + (a+b)^*$ $(a+b)^*$

Steps :- final R.E must be.

$$R_{12}^{(2)} = b^* a (a+b)^*.$$

This the equivalent R.E for the given DFA.

NOTE:- Simplifications

1. $\epsilon R = R\epsilon = R$
2. $\phi R = R\phi = \phi$
3. $(\epsilon)^* = \epsilon$ and $(\phi)^* = \epsilon$.
4. $\phi + R = R$.
5. $R + R = R$
6. $RR^* = R^*R = R^+$
7. $(R^*)^* = R$.
8. $\epsilon + RR^* = \epsilon + R^*R = R^*$.
9. $R^*R^* = R^*$
10. $R^* + \epsilon = R^*$
11. $(R+\epsilon)^* = R^*$
12. $(R+\epsilon)R^* = R^*(R+\epsilon) = R^*$.
13. $(R+\epsilon)(R+\epsilon)^*(R+\epsilon) = R^*$.
14. $R^*S + S = R^*S$., $SR^* + S = SR^*$.
15. $\phi + \epsilon = \epsilon$

NOTE;-

1. $R^* = \{\epsilon, R, RR, RRR, \ldots\}$
2. $RR^* = R\{\epsilon, R, RR, RRR, \ldots\}$
3. $R^+ = \{R, RR, RRR, \ldots\}$.

   $R^*(R+\epsilon)$

   $R^*R + R^*$

   $R^+ + R^*$.
4. $SR^* = \{S, SR, SRR \ldots\} \cup \{S\}$.
5. $R^*S = \{S, RS, RRS \ldots\} \cup \{S\}$.

3.2.2 Converting DFA's to Regular Expressions by Eliminating states.

The method for converting a DFA to R.E always Works. In fact, as you may have noticed, it doesn't really depend on the automaton being deterministic, and could be just as well NFA or even an ϵ-NFA.

The construction of the R.E is expensive. Not only do we have to construct about n^3 expressions for an n-state automaton, but the length of the expression can grow by a factor of 4 on average, with each of the n inductive steps. ($\because 4^n$ symbols)

There is a similar approach that avoids duplicating work at some points.

for eg, all the expressions with superscript $(k+1)$ in the construction of theorem 3.4 use the same subexpression $(R_{kk}^{(k)})^*$; the work of writing that expression is therefore repeated n^2 times.

The approach to constructing R.E that we shall now learn involves eliminating states.

When we eliminate a state s, all the paths that went through s no longer exist in the automaton. If the language of the automaton is not to change, we must include, on an arc that goes directly from q to p, the labels of paths that went from some state q to state p, through s.

Since the label of this arc may now involve strings, rather than single symbols, and there may even be an infinite number of such strings, we cannot simply list the strings as a label. Fortunately, there is a simple, finite way to represent all such strings: Use a R.E

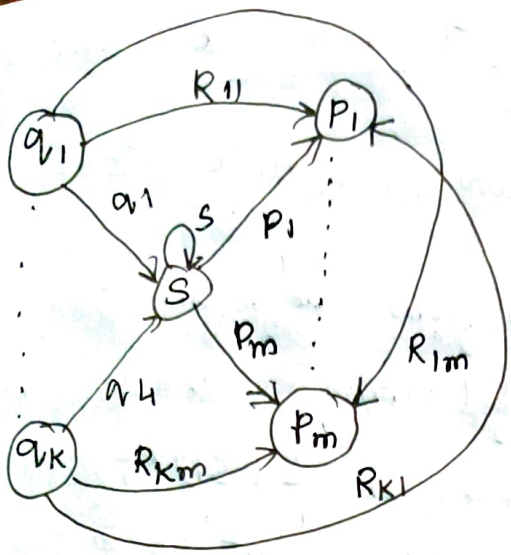


Figure 1: A state s about to be eliminated.

Figure 1, shows a generic state s about to be eliminated. We suppose that the automaton of which S is a state has predecessor states $q_1, q_2, \ldots, q_K$ for s and successor states $p_1, p_2, \ldots, p_m$ for s. It is possible that some of the q's are also p's, but we assume that s is not among the q's or p's, even if there is a loop from s to itself, as suggested by Figure 1.

Figure 2, shows what happens when we eliminate state s. All arc involving state s are deleted. To compensate, introduce, for each predecessor q_i of s and each successor p_j of s, a R.E that represents all the paths that start at q_i, go to s, perhaps loop around s zero or more times, and finally go to p_j.

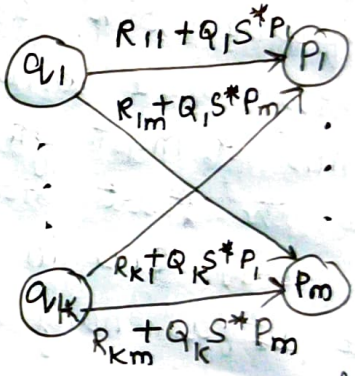


Figure 2: Result of eliminating state s from Figure 1.

The expression for these path is $Q_1S^*P_j$. This expression is added (with the Union operator) to the arc from q_i to p_j. If there was no arc $q_i \longrightarrow p_j$, then first introduce one with R.E ϕ.

The strategy for constructing a R.E from a finite automaton is as follows:

1. For each accepting state q, apply the above reduction process to produce an equivalent automaton with R.E labels on the arcs. Eliminate all states except q and the start state q_0.
2. If $q \neq q_0$, then we shall be left with a two-state automaton that looks like figure 3. The regular expression for the accepted strings can be described in various ways.

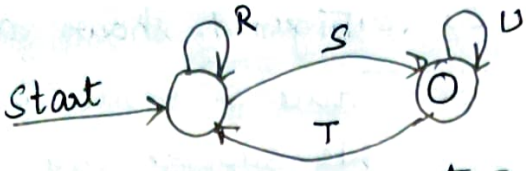


Figure 3. A generic two-state automaton.

One is $(R + SU^*T)^*SU^*$.

In explanation, we can go from the start state to itself any no. of times, by following a sequence of paths whose labels are in either $L(R)$ or $L(SU^*T)$. Expression SU^*T represents paths that go to the accepting state via a path in $L(S)$, perhaps returning to the accepting state several times using a sequence of paths with labels in $L(U)$, and then returning to the start state with a path whose label is in $L(T)$. Then we must go to the accepting state, never to return to the start state, by following a path with a label in $L(S)$. Once in the accepting state, we can return to it as many times as we like, by following a path whose label is in $L(U)$.

3. If the start state is also accepting state, then we must also perform a state-elimination from the original automaton that gets rid of every state but the start state. When we do so, we are left with a one-state automaton that looks like figure 4. The R.E denoting the strings that it accepts is R^*.

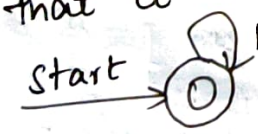


Figure 4. A generic one-state automaton.

4. The desired R.E is the sum (Union) of all the expressions derived from the reduced automata for each accepting state, by rules (2) and (3).

Example 1 :- Consider the NFA in figure 1. which accepts all strings containing 110.

Soln :- First convert it to an automaton with R.E labels.

This is achieved by replacing the label "0, 1" with equivalent R.E 0+1. The result is shown in figure. 2.

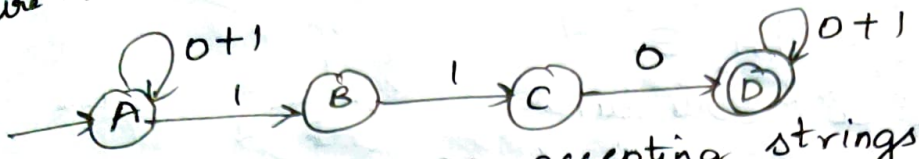


figure 1. An NFA accepting strings containing 110.

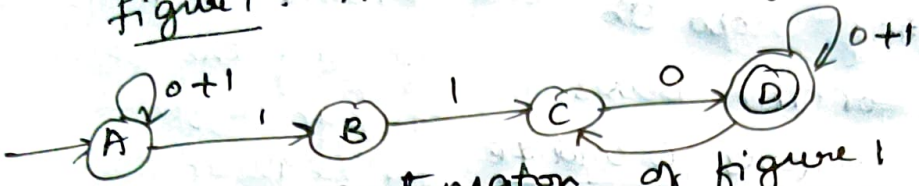


Figure 2: The automaton of figure 1 with R-E labels.

Since state B is neither an initial nor an accepting state it will not be in any of the reduced automata. Hence we choose to eliminate it first. Since there is no transition from A to C (i.e, $Q_1 = 1$, $P_1 = 1$, $R_{11} = \phi$), Thus the new transition from A to C and no loop at state B ($S = \phi$) Since $\phi^* = \epsilon$, we get 11 on simplification. The new automaton is shown in figure 3.

figure 3. Eliminating state B.

We now eliminate C in a similar manner and the result is shown in figure 4.

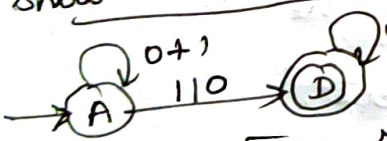


figure 4. A two-state automaton with states A and D.

In terms of the generic two-state automaton, ~~of figure~~. The R.E from figure. 4 are

$R = 0+1$, $S = 110$, $T = \phi$ and $U = 0+1$.

Since $T = \phi$, the expression SU^*T is also ϕ.

Thus the expression $(R + SU^*T)^*SU^*$ becomes

$(1+0)^* 110 (1+0)^*$.

3.2.3 Converting Regular Expressions to Automata

Figure shows that every language L is that L(R) for some R.E R, is also L(E) for some ϵ-NFA E.

The proof is a structural induction on the expression R.

Start by showing how to construct automata for the basis expressions: single symbols, ϵ and ϕ.

Then show how to combine these automata into larger automata that accept the union, concatenation, or closure of the language accepted by smaller automata.

All the automata we construct are ϵ-NFA's with a single accepting state.

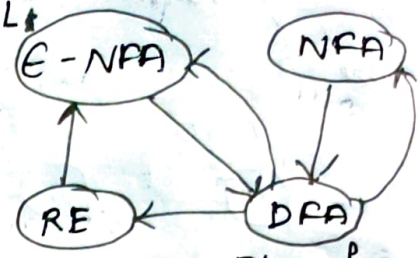


Figure. Plan for showing the equivalence of four different notations for Regular languages.

Theorem:-
Every language defined by a R.E is also defined by a finite automaton.

Proof:- Suppose L = L(R) for a R.E R. We S.T L = L(E) for some ϵ-NFA E with:

1. Exactly one accepting state.
2. No arcs into the initial state.
3. No arcs out of the accepting state.

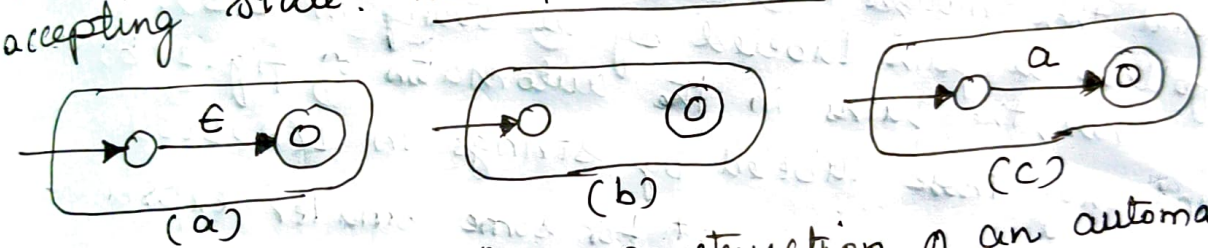
The proof is by structural induction on R, following the recursive definition of R.E that was discussed in section 3.1.2.

BASIS: There are three parts to the basis, shown in below figure.1. In part (a) see how to handle the expression ϵ. The language of the automaton is easily seen to be {ϵ}, since the only path from the start state to an accepting state is labeled ϵ.

Part (b) shows the construction for ϕ. Clearly there are no paths from start state to accepting state, so ϕ is the language of this automaton.

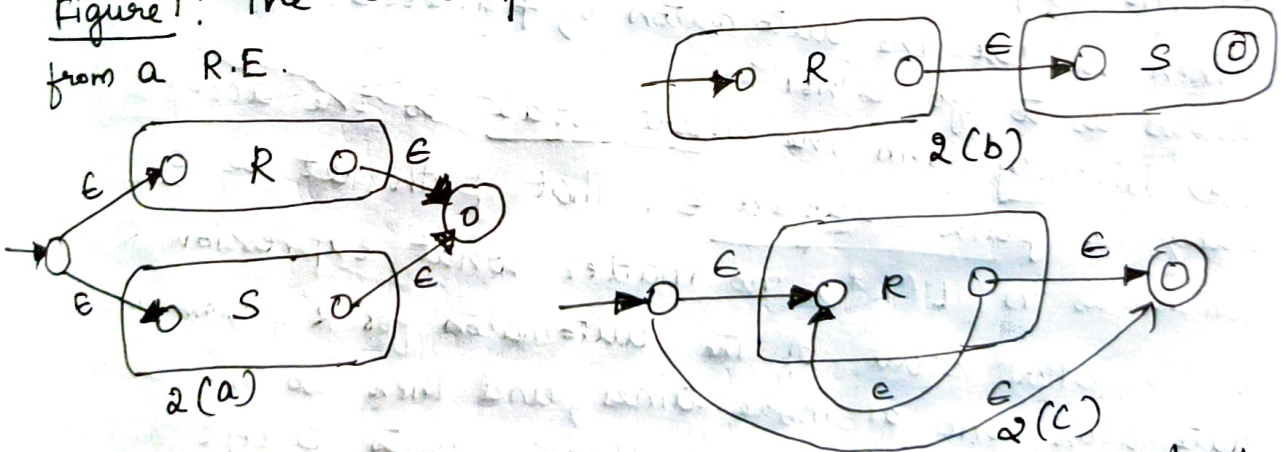
Finally, part (c) gives the automaton for R.E a. The language of this automaton evidently consists of the one string a, which is also L(a). It is easy to check that these automata all satisfy conditions (1), (2), and (3) of the inductive hypothesis.

INDUCTION: The three parts of the induction are shown in figure 2. Assume that the statement of the theorem is true for the immediate subexpressions of a given R.E; i.e, the languages of these subexpressions are also the languages of ϵ-NFA's with a single accepting state. The 4 cases are:-



(a) (b) (c)

Figure 1: The basis of the construction of an automaton from a R.E.



2(a) 2(b) 2(c)

Figure 2: The inductive step in the R.E to ϵ-NFA construction.

① The expression is R+S for some smaller expressions R and S. Then the automaton of fig 2(a) serves. i.e, Starting at the new start state, we can go to the start state of either the automaton for R or the automaton for S.

We then reach the accepting state of one of these automata, following a path labeled by some string in L(R) or L(S), respectively.

Once we reach the accepting state of the automaton for R or S, we can follow one of the ϵ-arcs to the accepting state of the new automaton.

Thus, the language of the automaton in fig 2(a) is L(R) ∪ L(S).

(2) The expression is RS for some smaller expressions R and S. The automaton for concatenation is shown in figure 2(b).

The idea is that only paths from start to accepting state go first through the automaton for R, where it must follow a path labeled by a string in L(R), and then through the automaton for S, where it follows a path labeled by a string in L(S).

Thus, the paths in the automaton of fig. 2(b) are all and only those labeled by strings in L(R)L(S).

(3) The expression is R^* for some smaller expression R. Then we use the automaton of fig. 2(c). That automaton allows us to go either:

(a) Directly from the start state to the accepting state along a path labeled ϵ. That path let us accept ϵ, which is in $L(R^*)$ no matter what expression R is.

(b) To start state of the automaton for R, through that automaton one or more times, and then to the accepting state. This set of paths allows 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 perphas ϵ, which was covered by the direct arc to the accepting state mentioned in 3(a).

(4). The expression is (R) for some smaller expression R. also serves as the automaton for (R), since the parentheses do not change the language defined by the expression.

It is a simple observation that the constructed automata satisfy the 3 conditions given in the inductive hypothesis — one accepting state, with no arcs into the initial state or out of the accepting state.

Example 1:-

Let us convert the R.E $(0+1)^*1(0+1)$ to an ϵ-NFA.

Soln:- Step1 :- first step is to construct an automaton for $0+1$.

(i.e two automata constructed according to figure 1(c), one with label 0 on the arc and one with label 1). (These two automata are then combined using the union construction of figure 2(a). The result is shown below.)

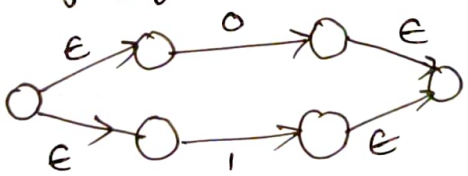


figure (a)

Step2 :- Next, apply to figure (a) the star construction of figure 2(c). This automaton is shown in figure (b).

The last two steps involve applying the concatenation construction of figure 2(b).

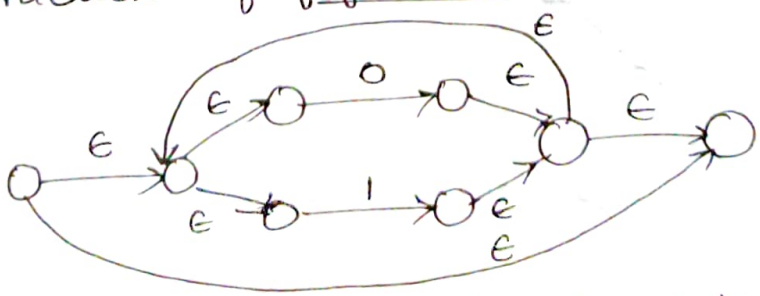


figure (b)

First, connect the automaton of figure (b) to another automaton designed to accept only the string 1.

Note that we must create a new automaton

to recognize 1; we must not use the automaton for 1 that was part of figure (a).

Step3 :- The third automaton in the concatenation is another automaton for 0+1.

Again, create a copy of the automaton of figure(a);

The complete automaton is shown in figure (c).

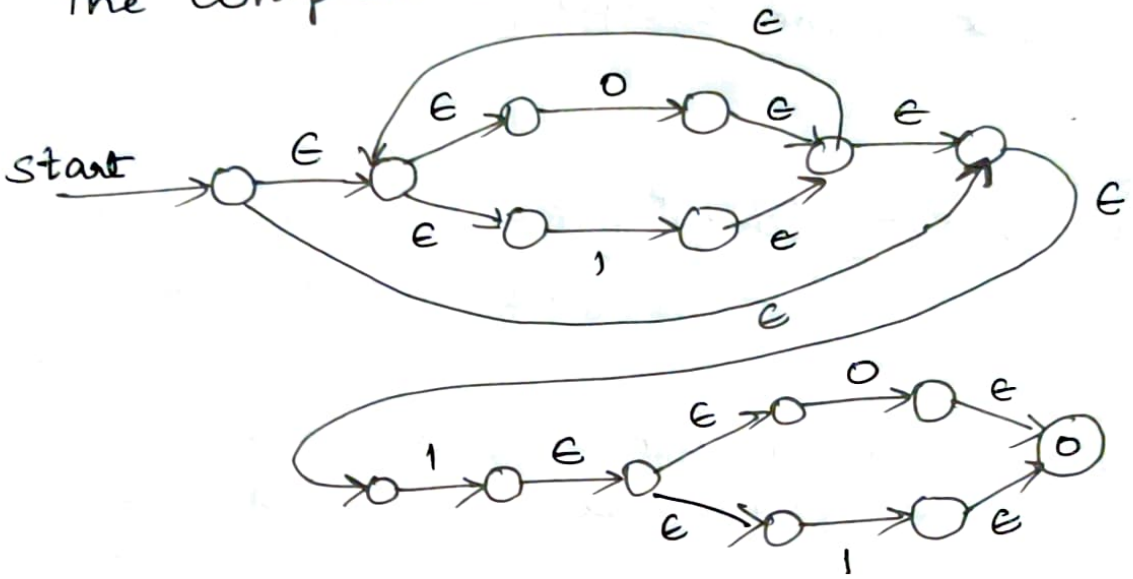


figure (c)