MODULE - 2

Prof TRM Rahman

Regular Expressions

Regular Expressions are short algebraic notations used to describe Regular Languages.

For example $(a+b)^*ab$ is a Regular Expression that describe the language consisting of set of all strings of a's and b's that end with ab.

Regular Expression	Language described by RE
$01(0+1)^*$	Set of all strings of 0's & 1's that begins with 01
$0(0+1)^*1$	Set of all strings that begin with 0 and end with 1
$(aa)^*$	Set of all strings Consisting of even number of a's
$(aa)^*a$	Set of all strings with odd number of a's
$(a+b)^*abb\ (a+b)^*$	Set of all string Containing Substring abb
$(a+b)^*abb$	Set of all strings that ends with abb

Operators of Regular Expressions:- In Regular Expressions we use three operators

(1) + (Union) (2) . (Concatination and (3) * (closure)

Scanned by CamScanner

Operations on the Languages:-

(1) Union:- Union of two languages L & M denoted by LUM is a new language that consist of strings chosen from either L or M or both.

Example: if $L = \{01, 110, 11\}$ $M = \{101, 00, 11\}$

then $L \cup M = \{01, 110, 11, 101, 00\}$

(2) Concatenation: Concatination of two languages L & M denoted by $L \cdot M$ is a new language obtained by taking any string from L and Concatinating it with any string in M.

Example: if $L = \{001, 10, 111\}$ $M = \{\epsilon, 001\}$

then $L \cdot M = \{001, 10, 111, 001001, 10001, 111001\}$

(3) closure: closure of a language L denoted by L^* is a set of those strings that can be formed by taking any number of strings from L, possibly with repetitions (i.e same string may be selected more than once). and Concatinating all of them.

Example: if $L = \{0, 1\}$

$L^* = \{\epsilon, 0, 1, 01, 10, 010, 101, 1101, 010 11101, \ldots\}$

If $L = \{0, 11\}$

then $L^* = \{011, 110, 01111, 11011, 0011, \ldots\}$

Note: Regular Language is a formal language for which there exists either Finite State Machine (DFSM/NDFSM) or Regular Expression, For example "Set of all strings that end with abb" is a Regular language.

Scanned by CamScanner

Building Regular Expressions: Normally we Construct Complex Regular expression starting from Smaller regular expressions. i.e. We Usually Combine Smaller expressions by means of R.E. operators, (Union (+), dot (.), and closure (*)) into larger expression Just like arithmetic operators have precedence, Smaller Regular expressions are grouped based on Priority (Precedence) of R.E. operators

NOTE: *(Closure) operator has Highest precedence followed by . (Concatination) followed by + (Union). Paranthesis is Used to Violate the Precedence of operators.

Consider the Regular expression $01^* + 10^*$

This R.E. describes set of all strings that have either a zero followed by any no of 1's or 1 followed by any number of 0's.

The above Regular Expression Can be grouped into:

$$\Big(\underbrace{(0 \cdot (1^*))}_{\{1\}^*} + \underbrace{(1) \cdot (0^*)}_{\{0\}^*} \Big)$$

$\{0\} \cdot \{\epsilon, 1, 11, 111, \ldots\}$ $\quad$ $\{1\}\ \{\epsilon, 0, 00, 000, \ldots\}$

$\{0, 01, 011, 0111, \ldots\}$ $\quad U \quad$ $\{1, 10, 100, 1000, \ldots\}$

$\{0, 01, 011, 0111, \ldots\}, 1, 10, 100, 1000, \ldots\}$

Therefore R.E. $01^* + 10^*$ describes the set of all strings of 0's and 1's that Consist of either 0 followed by any number of 1's or 1 followed by any number of 0's.

Scanned by CamScanner

Important!

Regular expression can be defined as per Basis & induction step as follows :-

(i) Basis :- ϵ, ϕ and a are Regular expressions describing the languages $\{\epsilon\}$, ϕ and $\{a\}$ respectively. That is $L(\epsilon) = \{\epsilon\}$, $L(\phi) = \phi$ and $L(a) = \{a\}$

(ii) Induction :- (i) If E and F are Regular Expressions then E+F is a R.E. denoting union of L(E) and L(F). That is $L(E+F) = L(E) \cup L(F)$.

(ii) If E & F are Regular Expressions then E.F or EF is a R.E. denoting Concatination of L(E) and L(F). That is $L(E.F) = L(E) . L(F)$.

(iii) If E is a R.E. then E^* is also a R.E. denoting closure of L(E). That is $L(E^*) = (L(E))^*$.

(iv) If E is a R.E., then (E) is also a R.E denoting the same language as E.

Problems

Design R.E's for the following languages :

(1) Set of all strings that end with either a or bb.

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

(2) Set of all strings with even number of 1's

$$(11)^*$$

(3) Set of all strings with odd number of a's

$$a(aa)^*$$

(4) Design Regular expression for the language

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

The above language is set of all strings consisting of even number of a's followed by odd number of b's

$\therefore$ the R.E. is $(aa)^* b(bb)^*$

Scanned by CamScanner

(5) Design Regular Expression for the following Language: (July 2016 exam)

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

Solution:- The sum of a's and b's is even in two cases

Case (I) When both m and n are even

Case (II) When both m and n are odd

Case (I): String must have ^even no. of a's followed by even no. of b's

The equivalent R.E is $R_1 = (aa)^* (bb)^*$

Case (II): String must have ^odd number of a's followed by odd number of b's

The equivalent R.E is $R_2 = a(aa)^* b(bb)^*$

The Resultant Regular expression is:

$$R = R_1 + R_2$$

$$\boxed{R = (aa)^*(bb)^* + a(aa)^* b(bb)^*}$$

(6) Design R.E for $L = \{a^n b^m : n \geq 4, m \leq 3\}$

The language is a set of all strings consisting of atleast four a's followed by atmost three b's.

The equivalent Regular Expression is:

$$\boxed{(aaaa)\, a^* (\epsilon + b + bb + bbb)}$$

(7) Design R.E. for the set of all strings containing atleast one a and atleast one b. (Jan 2010)

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

(8) Design R.E. for $L = \{w \mid |w| \bmod 3 = 0\}$ over $\Sigma = \{a, b\}$

The Language Consist of set of all strings whose length is a multiple of 3.

The equivalent Regular Expression is:

$$\boxed{((a+b)(a+b)(a+b))^*}$$

Scanned by CamScanner

(9) Design R.E. for the set of all strings whose 4th symbol from Right end is b.

(Jan 2010)

.... 4th 3rd 2nd 1st symbol

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

or

$(a+b)^*\ b\ (a+b)^3$

July 2011

(10) Design R.E. for $L = \{ a^{2n}\ b^{2n} \mid n \geq 0,\ m \geq 0 \}$

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

(11) Design R.E for the language. Consisting of set of all strings of a's b's and c's such that 4th symbol from right end is a and end with b

.. 4th 3rd 2nd 1st symbol

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

(12) Desegin R.E. for language

$L = \{ a^n\ b^m \mid n \leq 4,\ m \geq 2 \}$

Set of all strings with atmost four a's followed by atleast two b's.

The equivalent R.E. is:

$R = (\epsilon + a + aa + aaa + aaaa)\ bb\ b^*$

(13) Desegin R.E. for the set of all strings of 0's and 1's having no two Consecutive 0's.

Solution: whenever 0 occurs, it must be followed by 1 and there is no restriction on number of 1's. So The string Can have any Combination of 1's and 01's

The Partial Regular Expression is: $(1+01)^*$

The above R.E. does not describe strings that end with 0.

Scanned by CamScanner

Design R.E for following languages

1) Set of all strings of a's and b's that end with b.

(Easy one)

$(a+b)^* b$

or

$(a,b)^* b$

2) Set of all strings that contains a as 3rd character

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

or

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

---

3) $L = \{ w \in \{a,b\}^* : w$ contains odd number of a's $\}$

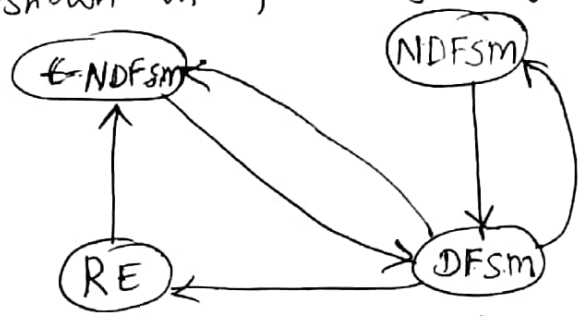
$b^*\left(a\, b^*\, a\, b^*\right)^* a\, b^*$

aa ... aa

The string may or may not end with 0. This is described by R.E. $(0 + \epsilon)$

The Final R.E. is $\boxed{(1 + 01)^* (0 + \epsilon)}$

Finite state Machine & Regular Expressions :- Both FSM and RE are notations used to describe Regular languages. There is a certain level of equivalence between these two notations, shown in following Figure.



The Figure shows equivalence between FSM and RE. That is

(i) Regular Expression R can be Converted to equivalent ε-NDFSM, E such that $L(R) = L(E) = L$

(ii) DFSM, A can be Converted to Regular Expression R such that $L(A) = L(R) = L$,

From Regular Expression to Finite state Machine :-

Theorem: Every language defined by a Regular Expression is also defined by a Finite state Machine

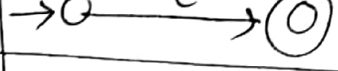
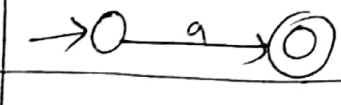
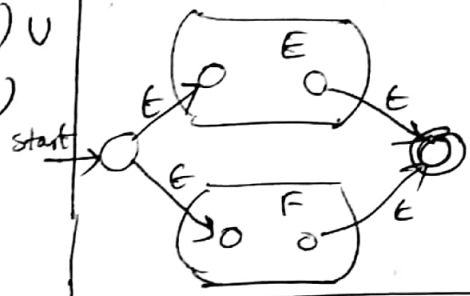
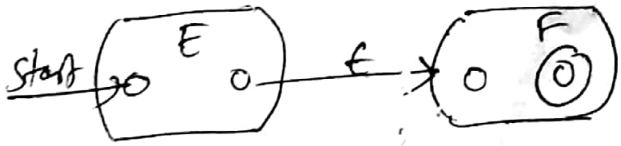
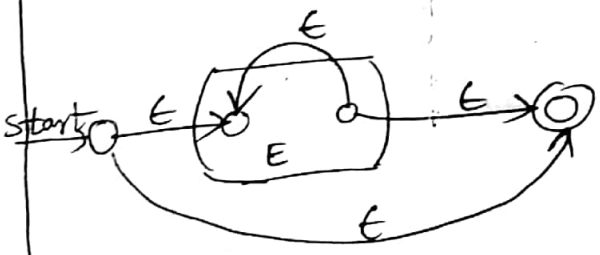
Proof :- Given a Regular Expression R. We need to show that every R.E. Can be Converted into equivalent ε-NDFSM E such that $L(R) \overset{RE}{=} L(E) = L$, and

ε-NDFSM E has ① Exactly one accepting state
② No arcs into the initial state
③ No arcs out of the accepting state.

Proof is by structural induction on R, as per the definition of R.E, following are Regular Expressions.

Scanned by CamScanner

ϵ, ϕ, a, $E+F$, $E \cdot F$, E^*. We need to show that there exist equivalent ϵ-NDFSm for each of the above Regular Expression. Following table illustrates this.

R.E.(R)	L(R)	Equivalent ϵ-NDFSm(E)	L(E)
ϵ	$L(\epsilon) = \{\epsilon\}$		$L(E) = \{\epsilon\}$
ϕ	$L(\phi) = \phi$		$L(E) = \phi$
a	$L(a) = \{a\}$		$L(E) = \{a\}$
$E+F$	$L(E+F) = L(E) \cup L(F)$		$L(E) = L(R)$
$E \cdot F$	$L(E \cdot F) = L(E) \cdot L(F)$		
E^*	$L(E^*) = (L(E))^*$		

Therefore it is shown that for every Regular Expression R there exists an equivalent ϵ-NDFSm E such that both R and E accept or describe the same language.

Scanned by CamScanner

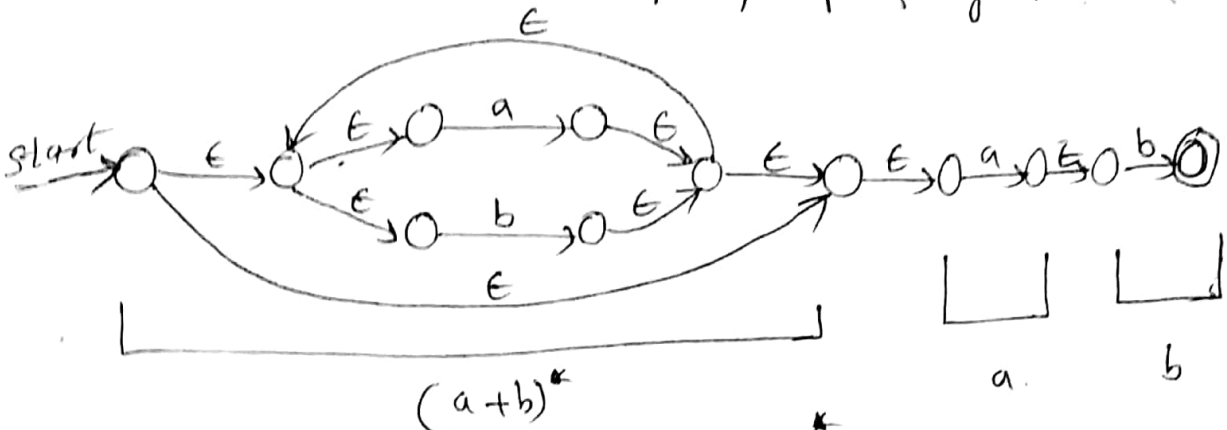
Problems:

(1) Convert the Regular expression $(a+b)^*.ab$ into its equivalent ϵ-NDFSM

The order of grouping of Smaller R.E.ns is:-

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

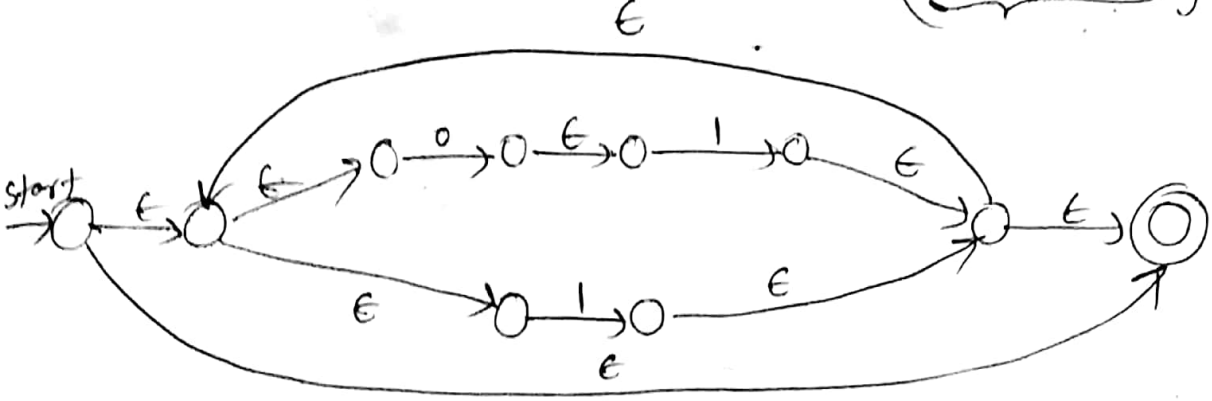
Design ϵ-NFA in the order of grouping of R.E.ns



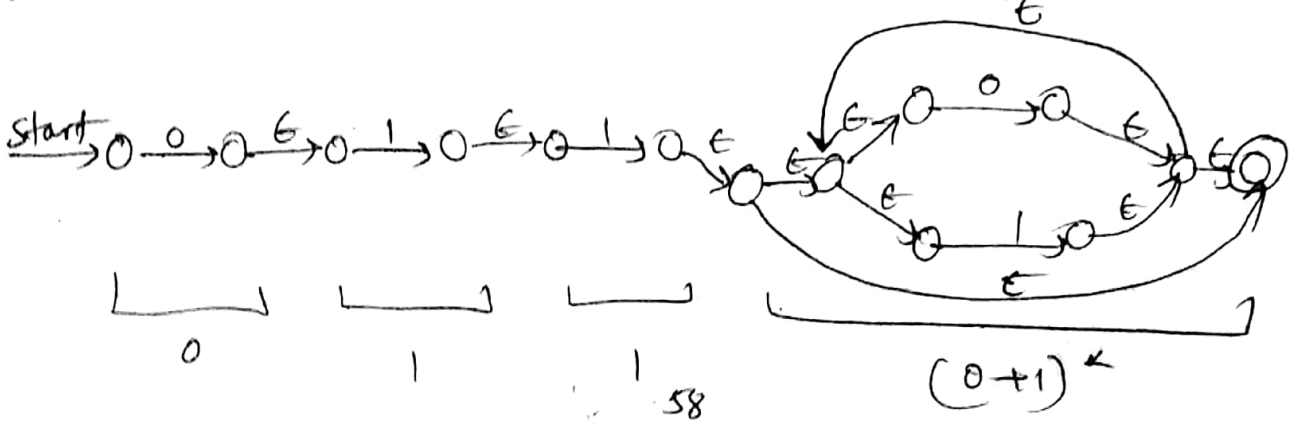
(2) Design ϵ-NDFSM for R.E $(01^* + 1)^*$

Note: within paranthesis Smaller R.E.ns are grouped Strictly based on the precedence of operators.

Here the order of grouping is $\left((0 \cdot 1) + 1\right)^*$

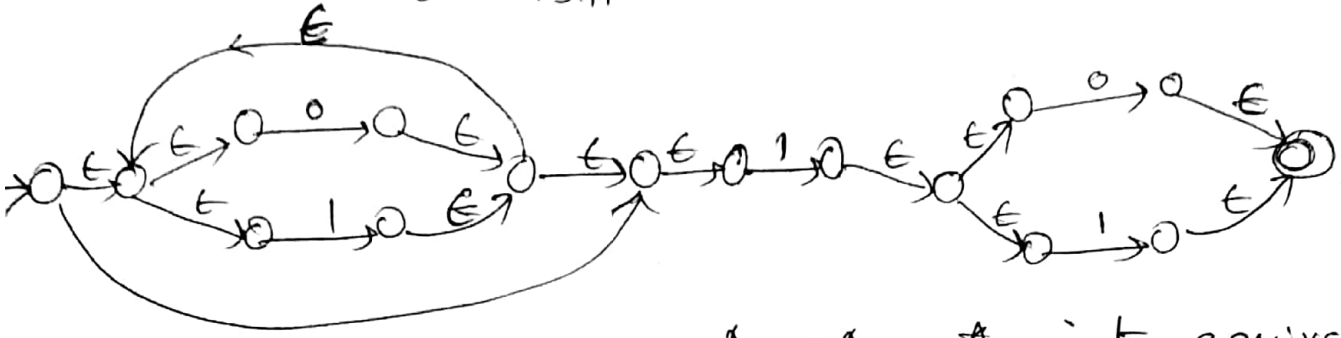


(3) Convert the R.E $011 \cdot (0+1)^*$ into ϵ-NDFSM

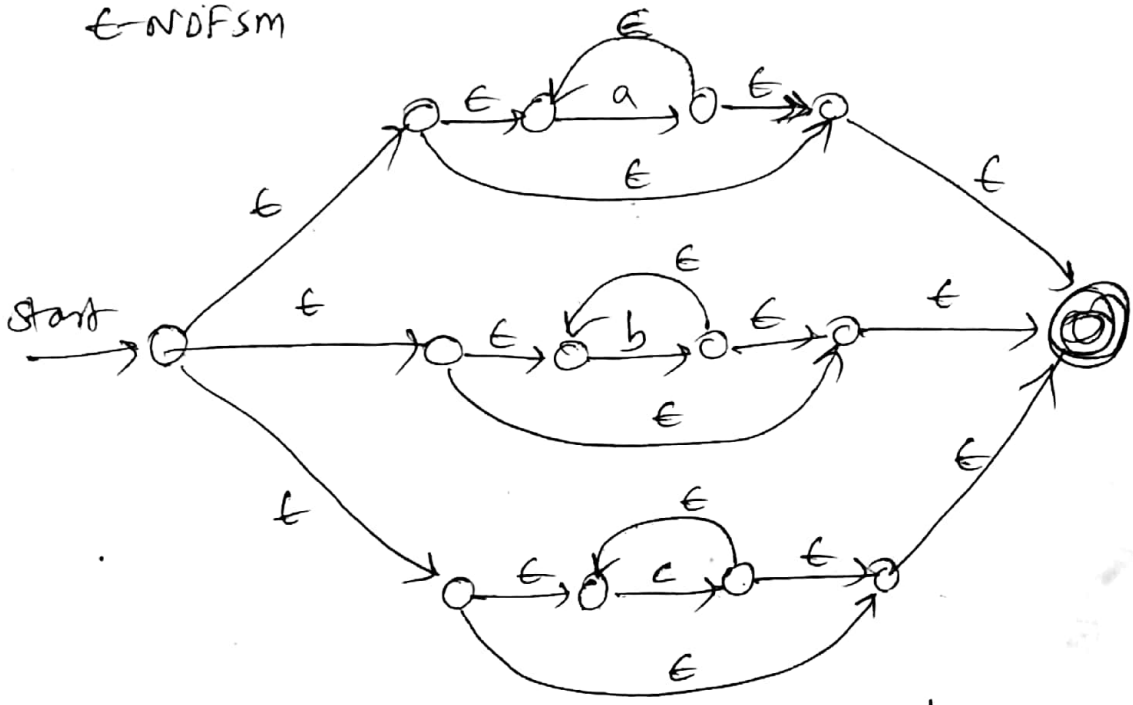


Scanned by CamScanner

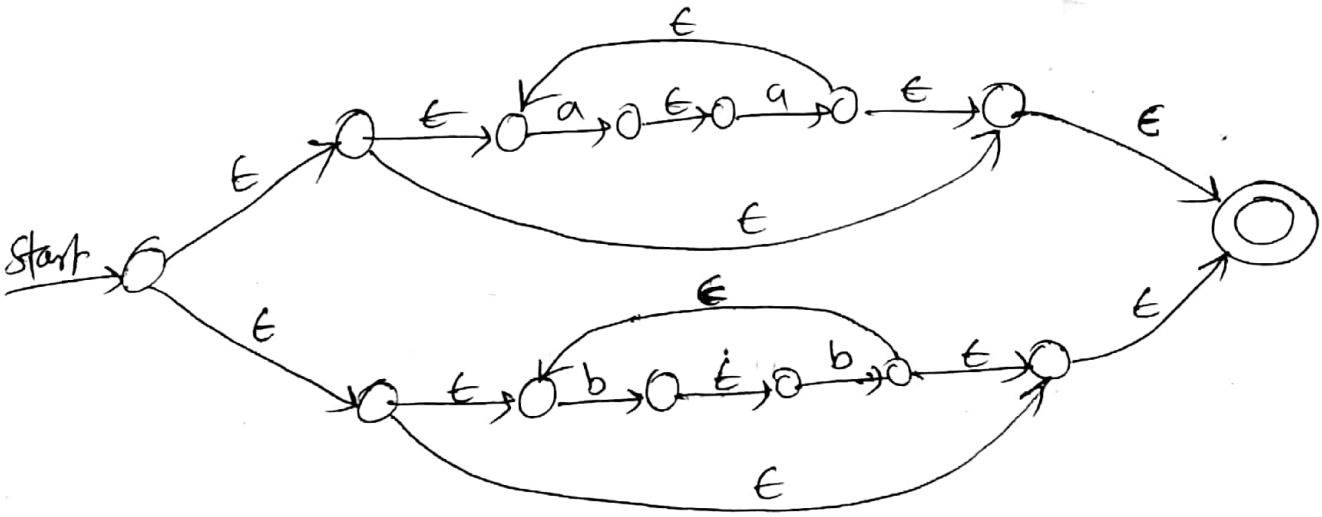
(4) Convert the R.E. $(0+1)^*1(0+1)$ into equivalent ϵ-NDFSM



(5) Convert the R.E. $a^* + b^* + c^*$ into equivalent ϵ-NDFSM



(6) Convert R.E. $(aa)^* + (bb)^*$ into ϵ-NDFSM



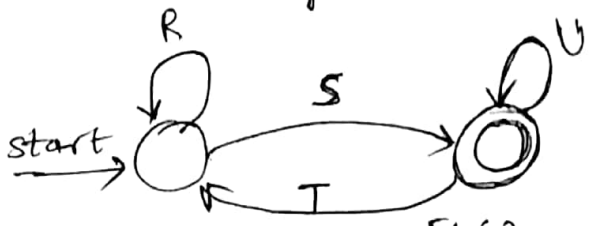
Scanned by CamScanner

Converting Finite State Machine to Regular Expressions

Using State elimination method :-

Procedure :-

1. Convert given DFSM, so that its labels are Regular Expressions
2. Eliminate trap state (dead state) if any and generate reduced FSM
3. Eliminate redundant states if any and generate reduced Automaton,
4. Finally, the [illegible] is reduced into one of the two generic Automata.



Fig(1)

or

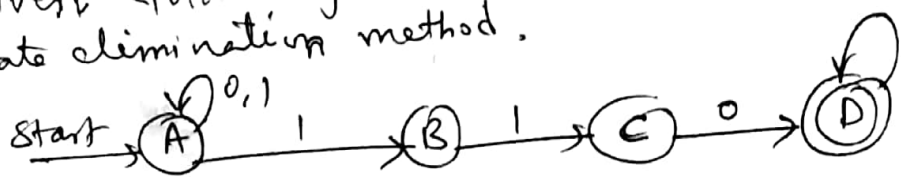
fig(2)

Equivalent R.E. is :- $(R+SU^*T)^*SU^*$

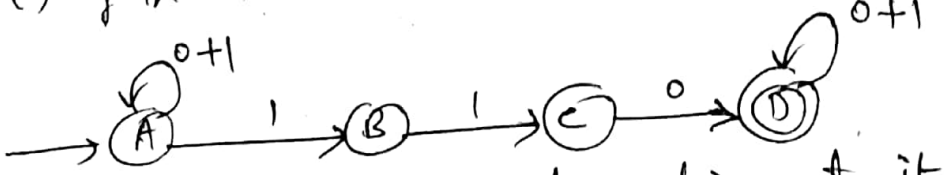
R.E is R^*

Problems :-

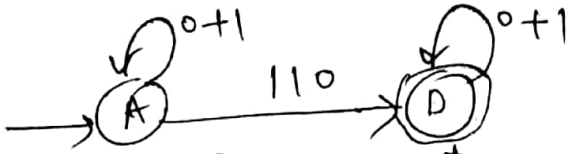
① Convert following NDFSM to Regular Expression using state elimination method.



apply step (1) of the Procedure



B & C are redundant states eliminate it



The automata is Reduced to a 2 state generic automata Fig(1).

$R = 0+1$ $S = 110$ $U = 0+1$

$$(R+SU^*T)^*SU^*$$

Substitute values $((0+1)+110(0+1)^*.\phi)^*110(0+1)^*$

Scanned by CamScanner

We know that $R \cdot \phi = \phi$ & $\phi \cdot R = \phi$,

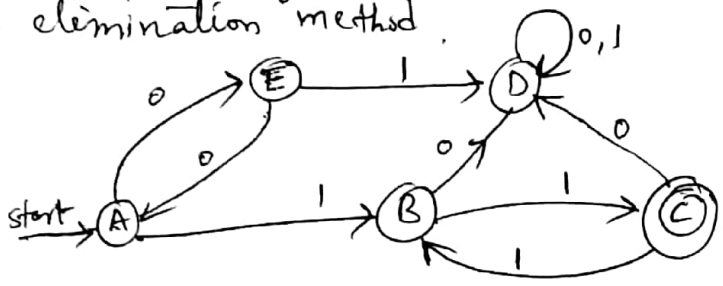
& $R + \phi = R$ & $\phi + R = R$

Where ϕ is a Regular Expression that describes empty language.

$$((0+1) + \phi)^* 110 (0+1)^*$$

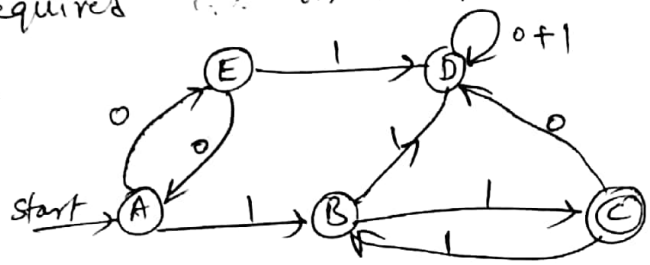
$$\boxed{(0+1)^* 110 (0+1)^*}$$

(2) Convert following DFSM to Regular Expression using state elimination method.



Note! Start and Final states can't be eliminated. They are required in order to accept a given string.

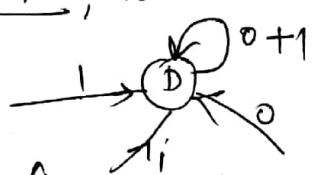
apply step (1):



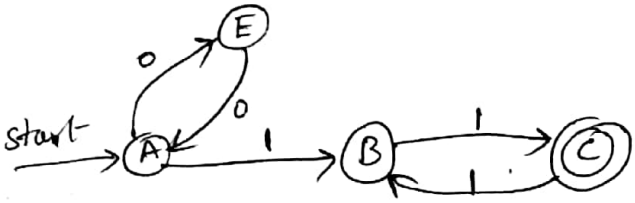
Eliminate state D as it is a trap state.

Note! trap state has only incoming arcs from other state if at all any outgoing arc, it is to itself.

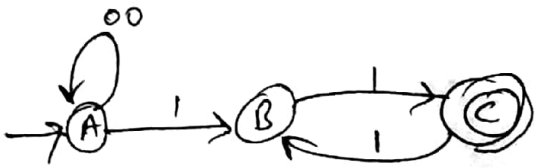
∴ D is a trap state:



(Trap state has only incoming arcs. there are no outgoing arcs)

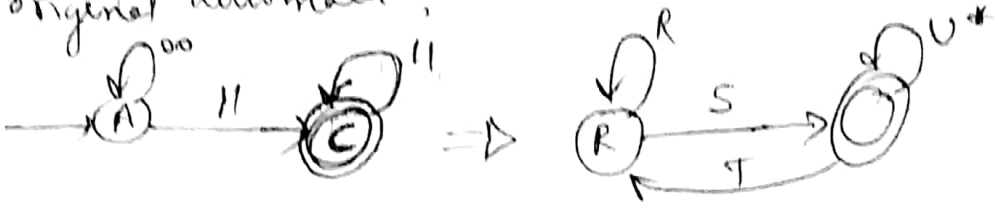


eliminate state E.



Scanned by CamScanner

Now eliminate State B preserving the language of the original automata.



$R = 00$
$S = 11$
$T = \phi$
$U = 11$

$(R + SU^*T)SU^*$

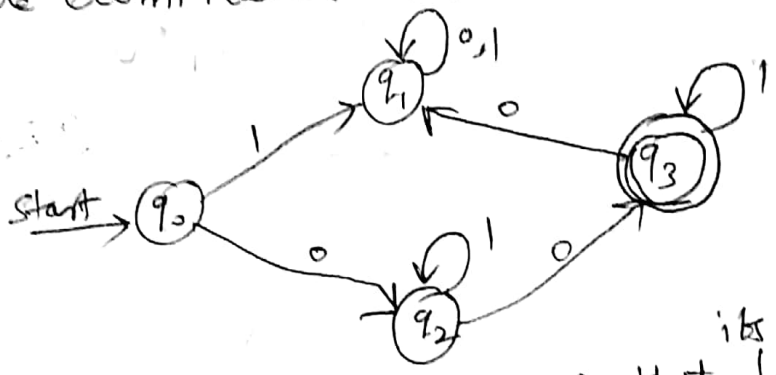
$((00) + 11(11)^*.\phi)^*.11(11)^*$

We know $R.\phi = \phi$

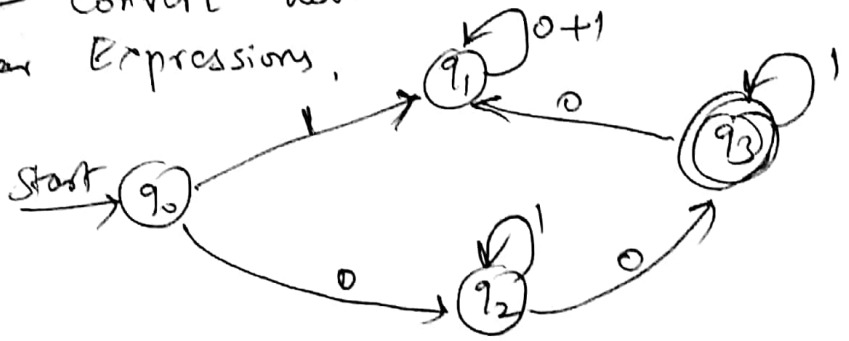
$(00)^* 11 (11)^*$

The Equivalent R.E. is :-

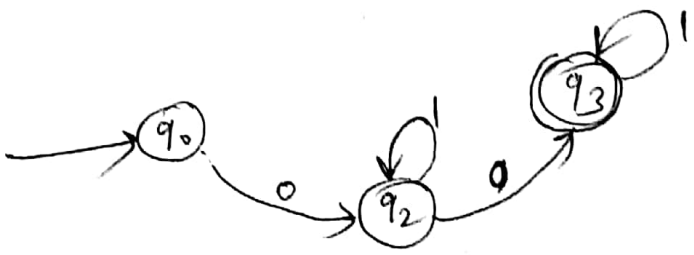
(3) obtain Regular Expression from the following NFA using state elimination method.



apply step ① :- Convert above NFA so that its labels are Regular Expressions.

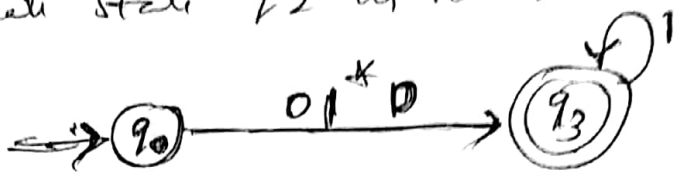


eliminate q_1 as it is a trap state

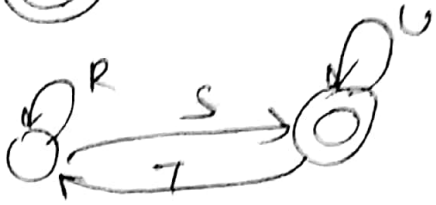


Scanned by CamScanner

eliminate state q_2 as it is redundant



general automaton is



$R = \phi$

$S = 01^*0$

$U = 1$

$(R + SU^*T)SU^*$

Substitute values $(\phi + 010^*1\phi)01^*0(1)^*$

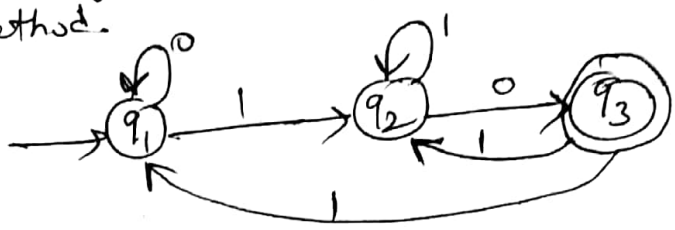
We know $R;\phi = \phi$

$(\phi + \phi)^*, 01^*0(1)^*$

$\epsilon \cdot R = R$

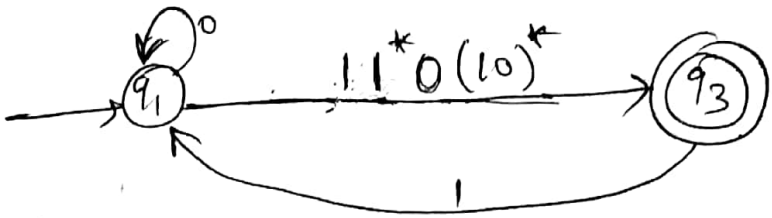
$01^*0(1)^*$

(4) Convert the following DFA to Regular Expression using state elimination method.

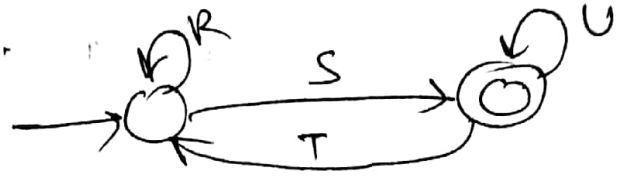


Convert given NFA so that labels are Regular Expressions resultant NFA is same as given NFA, Now labels are Regular Expressions.

Eliminate state q_2 as it is redundant



The given automaton is reduced to a two state generic automaton.



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

63

Scanned by CamScanner

$R = 0$ $\quad S = 11^*0(10)^*$ $\quad U = \phi$ $\quad T = 1$

$$\left(0 + \underbrace{11^*0(10)^*}_{S}\cdot\underbrace{(\phi)^*}_{U^*}\cdot\underbrace{1}_{T}\right)^*\underbrace{11^*0(10)^*}_{S}\underbrace{\phi^*}_{U^*}$$

We know $\phi^* = \epsilon$ & $R \cdot \epsilon = \epsilon \cdot R = R$

$$= (0 + 11^*0(10)^*\,\epsilon\cdot 1)^*\,11^*0(10)^*\cdot\epsilon$$

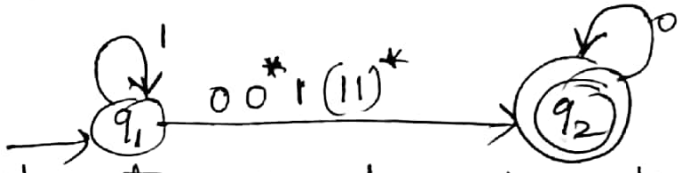
$$= \boxed{(0 + 11^*0(10)^*\,1)^*\,11^*0(10)^*}$$

Jan. 2014

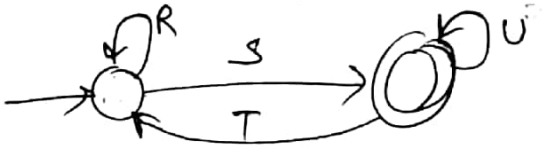
(5) Obtain Regular Expression from the following finite automaton Using state elimination method.

Convert above FA so that labels are Regular Expressions. The resultant automaton is same as given automaton. But now Labels are Regular Expressions.

Now Eliminate state q_2 as it is redundant



The Given automaton is reduced to a two state generic automaton



Whose R.E. is: $(R + SU^*T)^*SU^*$

$R = 1$
$S = 00^*(11)^*$
$T = \phi$
$U = 0$

$$\left(\underbrace{1}_{R} + \underbrace{00^*(11)^*}_{S}\underbrace{0^*}_{U^*}\cdot\underbrace{\phi}_{T}\right)^*\underbrace{00^*(11)^*}_{S}\underbrace{0^*}_{U^*}$$

We know $R \cdot \phi = \phi$

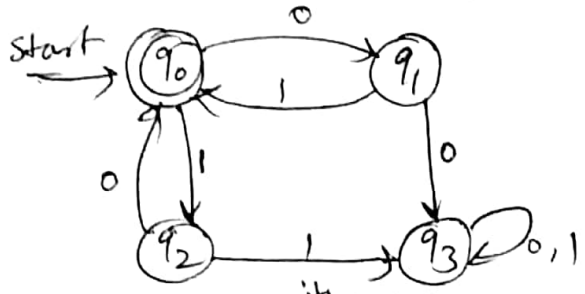
$$(1 + \phi)^*\cdot 00^*(11)^*0^*$$

We know $R + \phi = R$

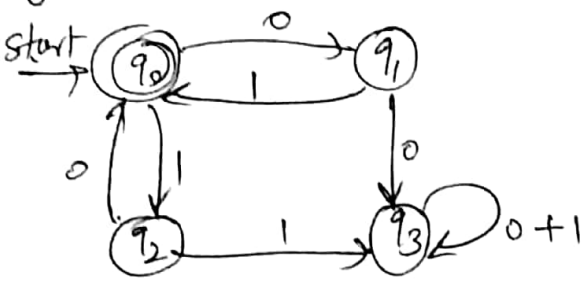
$$\boxed{1^*\cdot 00^*(11)^*0^*}$$

Scanned by CamScanner

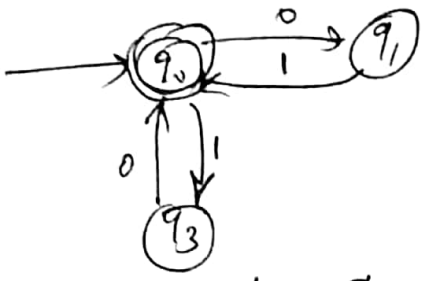
⑥ obtain Regular Expression for the FA of following figure.



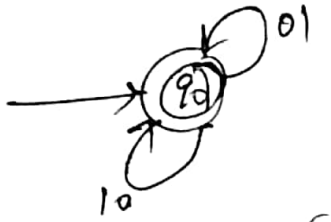
Convert given FA so that its labels are Regular expressions



eliminate State q_3 as it is a Trap state



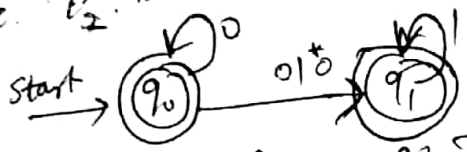
Now eliminate state q_1 & q_2 as they are redundant



equivalent R.E is $(01+10)^*$

⑦ obtain Regular Expression for the following Finite Automaton

State q_2 is a dead (trap state), Eliminate it



There are two final states.

for finite state q_0 the R.E. is $0^* = R_1$

——" —— q_1 the R E is $0^*11^* = R_2$

Scanned by CamScanner

$$R = R_1 + R_2$$

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

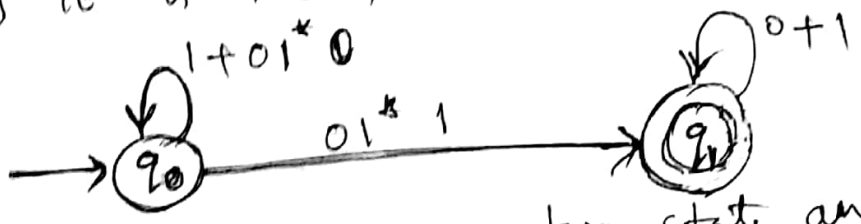
$$R = 0^* (\epsilon + 1 1^*)$$

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

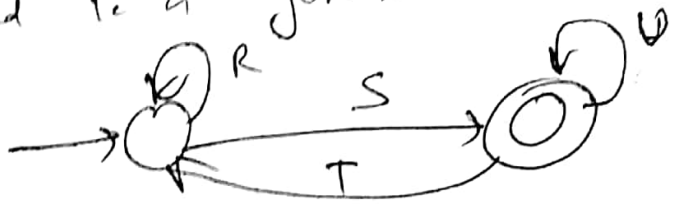
$$\boxed{R = 0^* 1^*}$$

(5) Convert following Finite Automata into Regular expression using state elimination method.

Label the arcs by regular expressions and eliminate state q_1 as it is redundant



FA is reduced to a generic two state automaton



$$R = (R + SU^* T)^* SU^*$$

$R = 1 + 01^* 0$
$S = 01^* 1$
$T = \phi$
$U = 0 + 1$

$$R = \left(\underbrace{(1 + 01^* 0)}_{R} + \underbrace{(01^* 1)}_{S} \underbrace{(0+1)^*}_{U^*} \cdot \underbrace{\phi}_{T} \right)^*$$

We know $R \cdot \phi = \phi$ & $R + \phi = R$

$\underbrace{01^* 1}_{S} \underbrace{(0+1)^*}_{U^*}$

$$(1 + 01^* 0 + \phi)^* \; 01^* 1 (0+1)^*$$

$$= \underline{(1 + 01^* 0)^* \; 01^* 1 (0+1)^*}$$

Scanned by CamScanner

From DFSM to Regular Expressions (Important).

It is possible to Convert a **given DFSM** into its equivalent Regular expression. Here we build a Collection of regular expressions that describe set of Strings that label Certain paths in transition Diagram of DFSM A.

example:

DFSM, A

Equivalent R.E

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

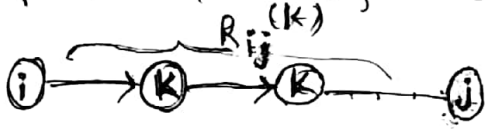
Kleen's theorem: (Imp).

If $L = L(A)$ for some DFSM A, then there is a Regular expression R such that $L = L(R)$

PROOF:- We need to prove that for every DFSM A, there is a Regular expression R such that $L(A) = L(R) = L$.

Let the DFSM has n states $\{1, 2, \ldots, n\}$.

Our first task is to Construct Collection of Regular expressions using DFSM A.



Let $R_{ij}^{(k)}$ is a R.E. that describes set of all strings w such that w is the label of a path from state i to state j and that path has intermediate nodes whose Values is less than or equal to k. (not higher than k).

Following figure shows requirement on the paths represented by $R_{ij}^{(k)}$

Fig: A Path whose label is in R.E. $R_{ij}^{(k)}$

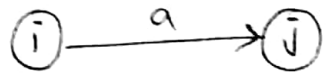
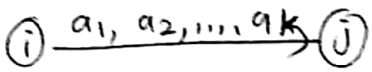
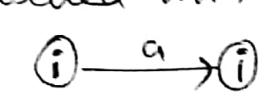
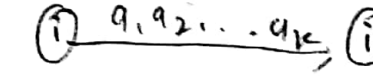
To Construct Regular expressions we use the following inductive definition Starting from $k=0$ to $k=n$.

BASIS: $k=0$ Since all states are numbered from 1 or above, the restriction on paths is that the path donot have intermediate state at all. (Since there is no state numbered with 0)

Scanned by CamScanner

There are only two types of paths that meet the condition. (1) An arc from state i to state j

$R_{ij}^{(0)}$ (2) An arc from state i to itself

Case	Type of Path	Possibilities	Equivalent R.E.
Case (1)	An arc from state i to state j.	(1) No Direct Path from state i to state j 	$R_{ij}^{(0)} = \phi$
		(2) An arc from state i to state j labelled with a 	$R_{ij}^{(0)} = a$
		(3) An arc from i to j labelled with $a_1, a_2, \ldots, a_k$. 	$R_{ij}^{(0)} = a_1 + a_2 + a_3 + \ldots a_k$
Case (2)	An arc from state i to itself	(1) Path of length 0 	$R_{ii}^{(0)} = \epsilon$
		(2) Path from i to i labelled with a 	$R_{ii}^{(0)} = \epsilon + a$
		(3) Path from state i to itself labelled with $a_1, a_2, \ldots, a_k$ 	$R_{ii}^{(0)} = \epsilon + a_1 + a_2 + \ldots + a_k$

Scanned by CamScanner

Induction:

Here, we construct Regular expressions $R_{ij}^{(k)}$ for $k = 1, 2, \ldots n$

Suppose there is a path from state i to state j passing through intermediate states whose values are less than or equal to k. There are 2 Possible ($k \geq 1$) Cases to Consider.

Case	Possible Cases	Equivalent R.E.
Case (1)	Path from i to j does not Pass through state k at all	$R_{ij}^{(k-1)}$
Case (2)	Path from i to j goes through state k atleast once. The Path can be broken into several pieces shown in following figure.	

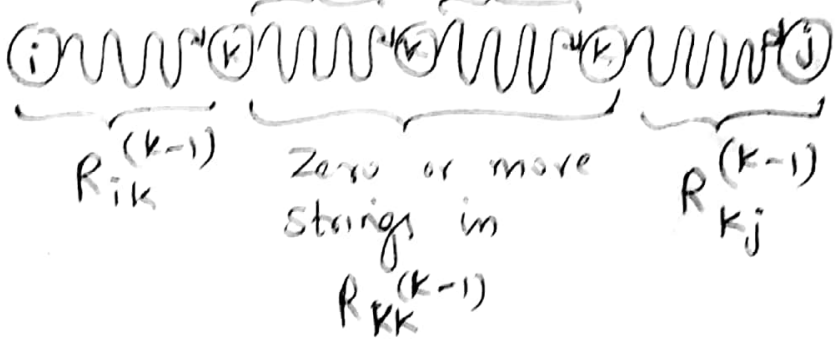


Fig:- Path from i to j Passing through intermediate states k.

The first path (from i to k) goes from state i to state k without Passing through k.

The middle Path go from state k to itself without Passing through k. Note that if the Path from i to j goes through state k only once, (i → k → j) then there are no middle pieces. Just a Path from i to j and a Path from k to j

Scanned by CamScanner

The last Path goes from state k to state j without passing through k.

All the three Paths from state i to state j are described by Regular expression

$$\underbrace{R_{ik}^{(k-1)}}_{1^{st}} \underbrace{\left(R_{kk}^{(k-1)}\right)^*}_{2^{nd}} \underbrace{R_{kj}^{(k-1)}}_{3^{rd}}$$

The first expression represents Part of the Path that goes to state k for the first time.

The Second expression represents Portion of Path that goes from k to itself zero one or more than once. The third expression represent part of the Path that leaves k the last time and goes to state j.

When we Combine the Paths of the two types (cases), we obtain the expression

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

for all Paths from i to j that go through intermediate states whose Value is <= k.

The Regular expression for the language is the Sum (Union) of all expressions $R_{ij}^{(n)}$ such that j is the accepting state and i is the initial state.

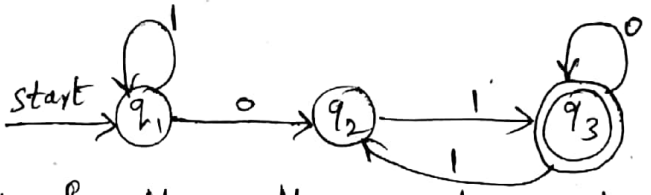
Scanned by CamScanner

for the labels for all paths from i to j that go through states whose values is less than or equal to k. If need to Construct these Regular expressions in the increasing order of subscript i.e $R_{11}^{(0)}$, $R_{12}^{(0)}$ ... $R_{21}^{(0)}$ ... $R_{nn}^{(0)}$, $R_{11}^{(1)}$ $R_{12}^{(1)}$ ..... $R_{ij}^{(n)}$.

We assume that state i & j are start and final states.

The R.E. for the language is the sum (Union) of all expressions $R_{ij}^{(n)}$ such that j is the accepting state.

Example 1 Convert the following DFSM to Regular Expression.



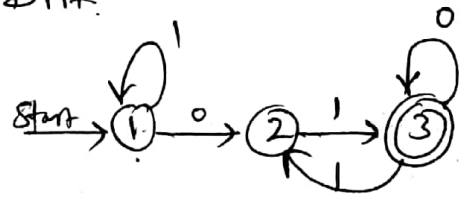
We Construct Regular Expressions for the paths that go through (i) no state ($k=0$) (ii) state 1 only ($k=1$) & (iii) states 1 and 2 only ($k \leq 2$)

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

Read as $R_{11}^{(0)}$, $R_{12}^{(1)}$, ...

In general $R_{ij}^{(k)}$.

Step ① :- Rename states q_1 q_2 q_3 into 1, 2 & 3 respectively. Construct DFA.



The above table gives Various stages of Construction, The basis expressions are shown in Second Column.

Scanned by CamScanner

$R_{ij}^{(1)}$ can be computed using $R_{ij}^{(0)}$ using

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

In general, $R_{ij}^{(k)}$ can be computed using $R_{ij}^{(k-1)}$

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

In Simplification use these inequalities:

$\epsilon + R = R$, $R + \epsilon = R$, $(\epsilon + R)^* = R^*$, $\phi \cdot R = R \cdot \phi = \phi$

The Final Regular expression is Computed as follows

$$R_{13}^{(3)} = R_{13}^{(2)} + R_{13}^{(2)} \cdot (R_{33}^{(2)})^* R_{33}^{(2)}$$

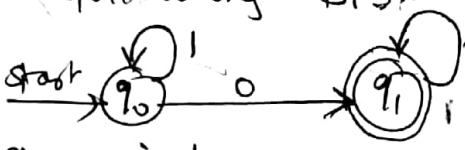
$i=1$ (start state)
$j=3$ (Final state)
$k=3$ passing through states whose values is $<=k$,
$n=3$
$\therefore k=3$

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

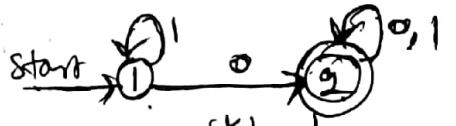
$$= 1^*01 (0 + 11)^*$$

$$\therefore \boxed{R_{13}^{(3)} = 1^*01 (0 + 11)^*}$$

Problem (2) :- (Important) Convert the following DFSM to its equivalent Regular expression



Step ①: Rename states q_0, q_1 into 1 and 2 respectively Construct DFA.



$n=2$ (no. of states) of FA

Construct Regular expressions $R_{ij}^{(k)}$ for $i=1$ to n, $j=1$ to n & $k=0$ to $n-1$

$R_{ij}^{(k)}$	$\leftarrow k \rightarrow$ (0)	(1)
R.E.Ns		
R_{11}	$\epsilon + 1$	1^*
R_{12}	0	1^*0
R_{21}	ϕ	ϕ
R_{22}	$\epsilon + 0 + 1$	$\epsilon + 0 + 1$

Read as $R_{11}^{(0)}$ $R_{11}^{(1)}$ $R_{12}^{(0)}$, $R_{12}^{(1)}$, $R_{21}^{(0)}$ etc.

Scanned by CamScanner

Compute $R_{ij}^{(k)}$ Using $R_{ij}^{(k-1)}$ for $k = 1$ to 2

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

$i=1$
$j=1$
$k=1$

$$R_{11}^{(1)} = R_{11}^{(0)} + R_{11}^{(0)} (R_{11}^{(0)})^* R_{11}^{(0)}$$

$$= (\epsilon+1) + (\epsilon+1)(\epsilon+1)^* \cdot \epsilon+1$$

$$= (\epsilon+1) + (\epsilon+1) 1^* (\epsilon+1)$$

$$= (\epsilon+1) + 1^*$$

$$\boxed{R_{11}^{(1)} = 1^*}$$

$i=1$
$j=2$
$k=1$

$$R_{12}^{(1)} = R_{12}^{(0)} + R_{11}^{(0)} (R_{11}^{(0)})^* R_{12}^{(0)}$$

$$= 0 + (\epsilon+1)(\epsilon+1)^* \cdot 0$$

$$= 0 + 1^* 0$$

$$\boxed{R_{12}^{(1)} = 1^* 0}$$

$i=2$
$j=1$
$k=1$

$$R_{21}^{(1)} = R_{21}^{(0)} + R_{21}^{(0)} (R_{11}^{(0)})^* R_{11}^{(0)}$$

$$= \phi + \phi \underbrace{(\epsilon+1)^* \cdot (\epsilon+1)}_{R}$$

$$\boxed{R_{21}^{(1)} = \phi}$$ Since $\phi \cdot R = \phi$

$i=2$
$j=2$
$k=1$

$$R_{22}^{(1)} = R_{22}^{(0)} + R_{21}^{(0)} (R_{11}^{(0)})^* R_{12}^{(0)}$$

$$= (\epsilon+0+1) + \phi (\epsilon+1)^* \cdot 0$$

$$\boxed{R_{22}^{(1)} = (\epsilon+0+1)}$$

Therefore, the Regular expression for given DFA A is -

$i=1$
$j=2$
$k=2$

$$R_{12}^{(2)} = R_{12}^{(1)} + R_{12}^{(1)} (R_{22}^{(1)})^* R_{22}^{(1)}$$

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

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

Scanned by CamScanner

Applications of Regular Expressions

Regular Expressions are used to develop following Softwares.

1. 'Lexical Analyzer', a Component of Compiler which reads characters of Source Program, and groups them into identifiers, numbers, keywords, operators, punctuation symbols, etc.
2. Web Search Engines which Search the World Wide Web for a given word, phrase, sentence or any other pattern.
3. Unix operating System to implement some of the Search Commands of Unix
4. perl interpreter which translates the perl program into object code.
   practical extraction and reporting language is often used to develop Server side applications.
5. Spam detection System to detect email messages in Spam
6. E-mail Search System to Search for given E-mail address from a text
7. Duplicate Word Search Software: To determine duplicate words present in the given text

Scanned by CamScanner

8. Search applications - to search a given database for a given pattern.

Manipulating and Simplifying Regular expressions :- Following identities are used to simplify given Regular expressions.

1. For any regular expressions α and β,
   $\alpha \cup \beta = \beta \cup \alpha$
   that is Union is Commutative
2. For any regular expressions α, β and γ,
   $(\alpha \cup \beta) \cup \gamma = \alpha \cup (\beta \cup \gamma)$.
   that is Union is associative
3. For any regular expression α,
   $\alpha \cup \phi = \phi \cup \alpha = \alpha$
   that is ϕ is identity for Union
4. For any regular expression α,
   $\alpha \cup \alpha = \alpha$
   that is Union is idempotent
5. For any regular expressions α, β, and γ,
   $(\alpha \cdot \beta) \cdot \gamma = \alpha \cdot (\beta \cdot \gamma)$
   that is Concatenation is associative,
6. For any regular expressions α,
   $\alpha \cdot \epsilon = \epsilon \cdot \alpha = \alpha$
   that is ϵ is identity for Concatenation,

Scanned by CamScanner

7. For any regular expression α,

$$\alpha \cdot \phi = \phi \cdot \alpha = \phi$$

that ϕ is zero for Concatination

8. For any regular expressions α, β, γ,

$$(\alpha \cup \beta) \cdot \gamma = \alpha\gamma \cup \beta\gamma$$

$$\gamma \cdot (\alpha \cup \beta) = \gamma\alpha \cup \gamma\beta$$

9. $\phi^* = \epsilon$

10. $\epsilon^* = \epsilon$

11. For any regular expression α, $(\alpha^*)^* = \alpha^*$

12 ——— 11 ——— $\alpha \cdot \alpha^* \cdot \alpha^* = \alpha^*$

13 For any R.E. α and β, $(\alpha \cup \beta)^* = (\alpha^* \beta^*)^*$

Important ?

The above Ways of Simplifying a Regular expression can be represented in short as follows:

Let $\alpha, \beta, \gamma, \epsilon, \phi$ be regular expressions

1	$\alpha \cup \beta = \beta \cup \alpha$	Union of R.Exs is Commutative
2	$(\alpha \cup \beta) \cup \gamma = \alpha \cup (\beta \cup \gamma)$	Union of R.E.s is associative
3	$\alpha \cup \phi = \phi \cup \alpha = \alpha$	ϕ is identity for Union
4	$\alpha \cup \alpha = \alpha$	Union of R.E.s is idempotent
5	$(\alpha \cdot \beta) \cdot \gamma = \alpha \cdot (\beta \cdot \gamma)$	Concatination of R.E.s is associative
6	$\alpha \cdot \epsilon = \epsilon \cdot \alpha = \alpha$	ϵ is identity for Concatination

Scanned by CamScanner

7	$\alpha \cdot \phi = \phi \cdot \alpha = \phi$	ϕ is zero for Concatination.
8	$(\alpha \cup \beta) \cdot \gamma = \alpha\gamma \cup \beta\gamma$	
9	$\phi^* = \epsilon$	closure of $R_1 \in \phi$ is same as $R_1 \in \epsilon$
10	$\epsilon^* = \epsilon$	
11	$\alpha \cdot (\alpha^*)^* = \alpha^*$	
12	$\alpha \cdot \alpha^* \cdot \alpha^* = \alpha^*$	
13	$(\alpha \cup \beta)^* = (\alpha^* \beta^*)^*$	

Scanned by CamScanner

Regular Grammers and Regular Languages :-

Finite state Machines & Regular Expressions are two equivalent ways to describe Regular Languages.

Regular Grammars :- Definition -

A Regular Grammar G is a quadruple (V, Σ, R, S) where V is a set of Variables or Non-Terminal Symbols. Each Variable generates its own language (set of strings)

Σ is a set of Terminals or Terminal Symbols. These terminal symbols together form strings.

R is a set of productions or Rules. Each Rule is of the form $X \longrightarrow Y$

In a Regular grammar, the Rules R must
* have a Left Hand Side that is a Single Non-Terminal
* have a Right Hand Side that is ϵ (epsilon) or a Single terminal or Single terminal followed by a Single Non-Terminal.

Terminals Symbols are lower case letters such as a, b, c, ..., digits such as 0, 1, 2, Operators such as +, -, *, etc punctuation symbols such are (,), {, }, ;, ,

Scanned by CamScanner

Non-Terminal Symbols are Capital letters such as A, B, C, ... S.

$S \to a$, $A \to b$, $B \to \epsilon$, $T \to aS$, $B \to bA$ are legal rules (productions) of grammer.

$S \to aSa$, and $aSa \to T$ are not legal rules in a regular grammer.

The Language generated by grammar G denoted by $L(G)$ is the set of all strings w in Σ^* such that it is possible to start with S and apply some finite set of rules in R and derive string w.

example: Consider the Regular Language:

$L = \{ w \in \{a,b\}^* : |w| \text{ is even} \}$

Regular Expression

$(aa + ab + ba + bb)^*$

DFSM

Regular grammer

$S \to \epsilon$ /* Stops derivation */

$S \to aT$ /* deriving 1st symbol of w */

$S \to bT$ /* -u- 2nd */

$T \to aS$, $T \to bS$ } /* derives remaining Symb */

$L = \{ ab, ba, abab, baba, aabb, bbaa, abbaaa, bbabab, \ldots \ldots \}$

Regular Language :- is a formal language for which there exist either Regular expression, or FSM or Regular grammer.

Derivation of string $w = abba$ for using above grammer:

$S \Rightarrow aT \Rightarrow abS \Rightarrow abbT \Rightarrow abbaS \Rightarrow abba$

$(S \to aT)$ $(T \to bS)$ $(S \to bT)$ $(T \to aS)$ $(S \to \epsilon)$

Scanned by CamScanner

Theorem: Regular Grammars define exactly the Regular language

Proof:- We know that For every Regular language, there exist equivalent FSM. To show that Regular Grammar define Regular language, We need to show that for every regular grammar there is an equivalent FSM.

The following algorithm Constructs FSM from a Regular grammar

$G = (V, \Sigma, R, S)$ Such that $L(M) \geq L(G)$.

grammartofsm (G: regular grammar) =

1. Create a Seperate state for each Non-Terminal Symbol of grammar.
2. Designate the start Non-Terminal of grammar as start state of FSM.
3. If there is any rule of the form $X \to a$ for some terminal Symbol a in Σ then Create an additional state labelled #.
4. For each rule of the form $X \to aY$ add a Transistion from X to Y labelled With Symbol a.

Scanned by CamScanner

5. For each rule of the form $X \to a$
add the transition from X to # labelled with a

6. For each rule of the form $X \to \epsilon$,
mark state X as accepting i.e. (X)

7. mark state # as accepting

Since every Grammar can be Converted into equivalent FSM and every FSM accepts regular language, Regular grammar define regular language.

Problems:-

(1) Design Regular grammar for the following Language:

$L = \{ w \in \{a, b\}^* : w$ ends with aaaa$\}$

The language contains set of all strings w that ends with aaaa. i.e. $L = \{$ abbaaaaa, babbaaaa, ... $\}$

The Regular expression for L is $(a+b)^* aaaa$

The Regular Grammar for L is Can be designed as follows.

$S \to aS$ ✓
$S \to bS$ ✓ } generates any number of a's & b's that is $(a+b)^*$

$S \to aA$ ✓ generate first a of the pattern aaaa
$A \to aB$ generate Second a " "
$B \to aC$ generate third a of the pattern aaaa.
$C \to a$ generate fourth a of the pattern aaaa.

Scanned by CamScanner

The grammer for given language is

$G=(V, \Sigma, R, S)$ where $V=\{S, A, B, C\}$

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

$R=\{S \to aS \mid bS \mid aA$

$A \to aB$

$B \to aC$

$C \to a$

$\}$

$S \to S$ (Start Variable)

applying grammer to fsm( ) to this grammer, we get the following FSM:

(2) Design Regular grammar for the following language: Assume $\Sigma=\{a, b, c\}$

$L = \{w$: there is a symbol '$\underline{a}$' $\in \Sigma$ not appearing in $w\}$

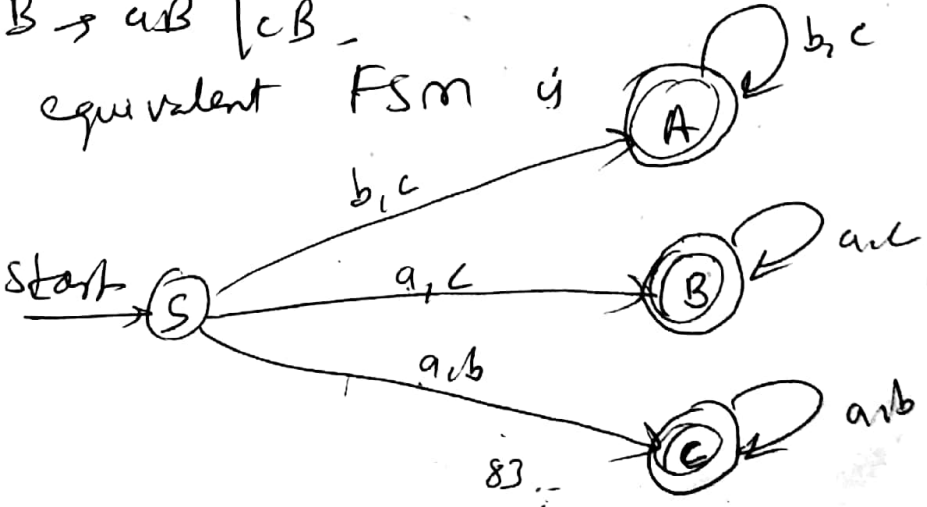
$S \to \in \mid aB \mid aC \mid bA \mid bC \mid cA \mid cB$

$A \to bA \mid cA \mid \in$ $\quad C \to aC \mid bC$

$B \to aB \mid cB$

Ex: $L=\{abab, acac, bcbc, \ldots\}$

the equivalent FSM is



Scanned by CamScanner

(3) Design Regular grammar for the following language

$L = \{ w \in \{a,b\}^* : w$ Contains odd number of a's and w ends in a $\}$

Solutions $L = \{bababa, abbaba, \ldots\}$

$S \to bS$ /* initial b's don't matter */

$S \to aT$ } /* after this no. of a's is odd and your string ends with a */

$T \to \in$ } /* stop derivation */

$T \to aS$ /* no. of a's are even */

$T \to bX$ /* no. of a's still odd and string donot end with a */

$X \to aS$ /* no of a's is even */

$X \to bX$ /* no. of a's is odd and string does not end with a */

$\therefore G = (V, T, P, S)$

Where $V = \{S, T, X\}$

$T = \{a, b\}$

$P = \{ S \to bS \mid aT$

$T \to \in \mid aS \mid bX$

$X \to aS \mid bX \}$

$S = S$ (Start Variable)

Scanned by CamScanner

Derivation of string $w = baaba$

Note: string contains odd no. of a's and ends with a.

$S \Rightarrow bS$ (S → bS) /* generates string with b as first symbol,

$\Rightarrow baT$ (S → aT) /* Now odd no. of a's & ends with a */

$\Rightarrow baaS$ (T → aS) /* Now even no. of a's */

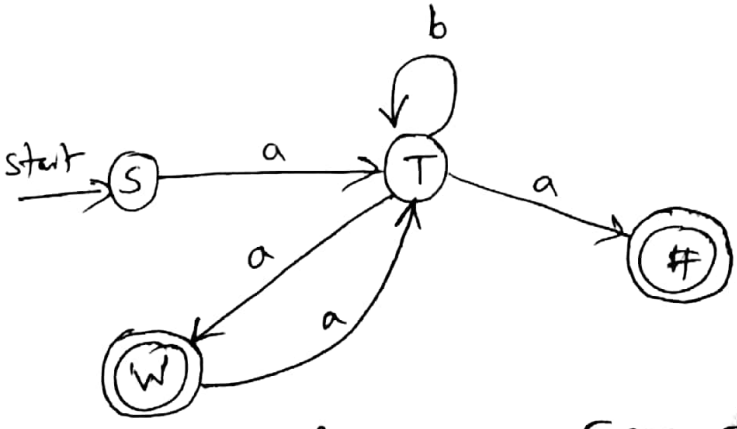
$\Rightarrow baabS$ (S → bS) /* still even no. of a's. */

$\Rightarrow baabaT$ (S → aT) /* Now odd no. of a's and ends with a */

$\Rightarrow baaba$ (T → ∈) /* stop derivation */

(4) Convert the following grammar to FSM using grammar to fsm ( )

$S \rightarrow aT$
$T \rightarrow bT$
$T \rightarrow a$
$T \rightarrow aW$
$W \rightarrow \epsilon$
$W \rightarrow aT$



(5) show by Construction that for every FSM there exists a regular grammar G such that $L(G) = L(M)$.

1. Make M deterministic (to get rid of ∈-transitions)
2. Create a Non-Terminal for each state in new M
3. Start state becomes start Non-Terminal
4. For each transition $\delta(T, a)$ make a rule of the form $T \rightarrow aU$

Scanned by CamScanner

5. For each accepting state T, make a rule of the form $T \to q$.

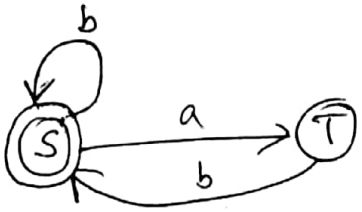
(6) Let $L = \{ w \in \{a,b\}^* :$ every a in w is immediately followed by atleast one b.

(i) Write regular Expression that describes L
(ii) Write a regular grammar that generates L
(iii) Write Fsm that accepts L.

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

(ii) R: $\{ S \to bS \quad V = \{S, T\}, \quad T = \{a, b\} \quad S = S$ (start Non-Terminal
$S \to aT$
$S \to \epsilon$
$T \to bS \}$

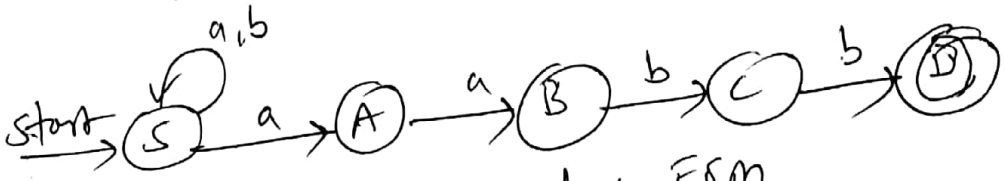
(iii)



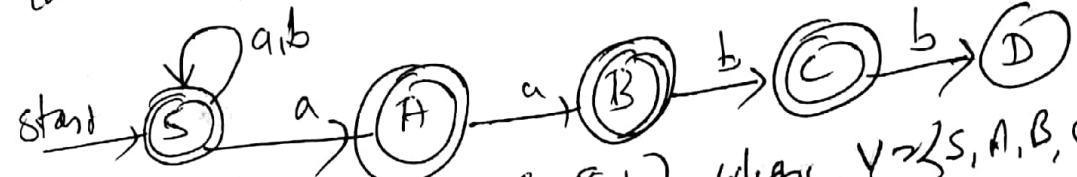
(7) Design grammar for $L = \{ w \in \{a,b\}^* :$ w does not end with aabb$\}$

$S \to aA$	$A \to aB$	$B \to aB$	$C \to aA$
$S \to bS$	$A \to bS$	$B \to bC$	$C \to \epsilon$
$S \to \epsilon$	$A \to \epsilon$	$B \to \epsilon$	

Step ①: Design Fsm for $L = \{ w \in \{a,b\}^* :$ w ends with aabb$\}$



take Compliment of above FSM.



Now Regular grammar $G = (V, \Sigma, R)$, where $V = \{S, A, B, C, D\}$, $\Sigma = \{a,b\}$, $R = \{ S \to aS \mid bS \mid aA, \; A \to aB, \; B \to bC, \; C \to b, S \to \epsilon, \; A \to \epsilon, \; B \to \epsilon, \; C \to \epsilon \}$

Scanned by CamScanner

Regular and Non-Regular Languages (RL) and (NRL).

Language	type	Reason
$L = \{ a^n b^n : n \geq 0 \}$ or $L = \{ \epsilon, ab, aabb, aaabbb, \ldots aaaabbbb, \ldots \}$ Note: Language has infinite no. of strings	Non-Regular Language	DFSM does not exist to accept string w Containing infinite no. of a's & b's. Note: DFSM Contains finite No. of states
$L = \{ w \in \{a, b\}^* : w$ ends with ba$\}$	Regular Language	DFSM exist to accept the language which Contains finite no. of strings

How many Regular Languages are there?

Theorem: The Regular Languages are Countably infinite.

Proof: We Can Construct DFSM's for each of the infinite no. of regular languages, There Can't be more regular languages than DFSM's. To accept infinite nu of regular languages, the DFSM's Can also be Countably infinite.
Hence the theorem.

Scanned by CamScanner

Showing that a language is regular

How many languages are regular -? This question can be answered by the following theorem:

Theorem: Every finite language is regular;

Finite language: is a language containing finite no. of strings.

Proof: If L is empty set, then it is defined by the Regular expression ϕ and so is regular. If L is a finite language consisting of finite no. of strings $S_1, S_2, \ldots S_n$ for some positive integer n, then it is defined by the Regular expression $S_1 \cup S_2 \cup \ldots \cup S_n$.

So it is also regular.

Closure Properties of Regular Languages:-

1. If L_1 & L_2 are two regular languages;

   $L_1 \cup L_2$ is also regular,

   $L_1 \circ L_2$ is also regular

   $L_1 \cap L_2$ is also regular

   $L_1 - L_2$ is also regular

That is Union of two regular languages is regular

Concatenation of ——— 11 ——— 11 ———

ϕ

Scanned by CamScanner

Intersection of two regular languages is regular

difference of ——" ——

2. If L is a regular language

L^* is also regular

L^R is also regular

h(L) is also regular.

That is closure of a regular language is regular.

Reverse of a ——" ——

Homomorphism of a regular language is regular.
(Letter substitution)

Theorem: The Regular languages are closed under Compliment, intersection, difference, reverse and letter substitution.

Proof:-

Case (1): Regular languages are closed under Compliment.

i.e. If L_1 is regular, $\overline{L_1}$ is also regular

If Since L_1 is regular, there exists a DFSM $M_1 = (Q_1, \Sigma, \delta_1, q_{0_1}, F_1)$ that accept L_1.

the DFSM $M_2 = (Q_2, \Sigma, \delta_2, q_{0_2}, F_2)$ where $F_2 = Q_1 - F_1$. From M_1 we can Construct M_2 by making Final states of M_1 as Non-Final states of M_2 and Non-Final states of M_1 as Final states of M_2.

Scanned by CamScanner

DFSM
Definitely M_2 accepts Compliment of L_1 which is $\bar{L}_1$. Therefore $\bar{L}_1$ is also regular.

Case(2): Regular Languages are closed under intersection.

i.e. If L_1 & L_2 are Regular languages, $L_1 \cap L_2$ is also regular

Proof: Since L_1 & L_2 are regular languages, there exists DFSM's $M_1 = (Q_1, \Sigma, \delta_1, q_{0_1}, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_{0_2}, F_2)$

For $L_1 \cap L_2$ we must Construct a DFSM,

DFSM M:-

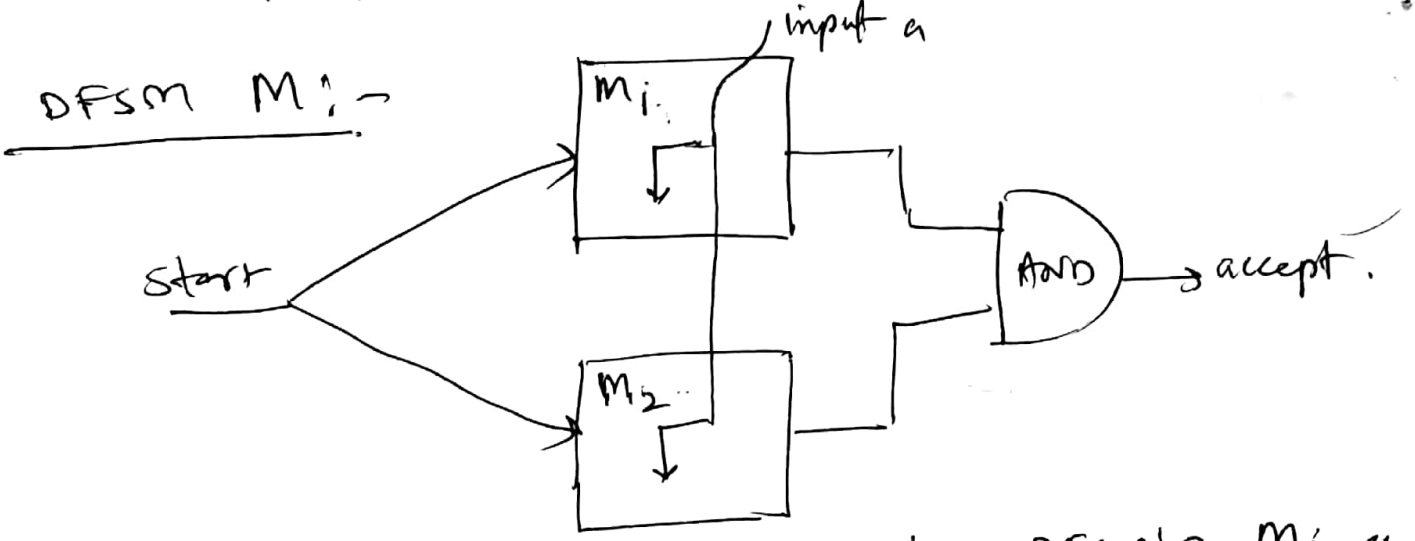


Fig:- A Single DFSM Simulating two DFSM's M_{L_1} & M_{L_2}

The DFSM M must be:

$M = (Q_1 \times Q_2, \Sigma, \delta, (q_{0_1}, q_{0_2}), F_1 \times F_2)$

Each state of M are pair of states (p, q) where state p is in M_1 & state q is in M_2.

Alphabet Σ of M is same as that of M_1 & M_2.

Scanned by CamScanner

δ (delta) of M can be defined as

$$\delta((p,q),a) = (\delta_1(p,a), \delta_2(q,a))$$

$q_0 = (q_{01}, q_{02})$ – i.e. start state of M is a pair of start states of M_1 & M_2 respectively.

(Start state of M)

$F \equiv F_1 \times F_2$ that is accepting states of are those pair of states (p,q) such that p & q are accepting states of M_1 & M_2 respectively.

(Final states of M)

DFSM M accepts w iff $\hat{\delta}((q_0,q_1), w)$ is a pair of accepting states, that is $\hat{\delta}(q_0, w)$ must be in F_1 & $\hat{\delta}(q_1, w)$ must be in F_2

Here $\hat{\delta}((\underbrace{q_0,q_1}_{\text{start state of M}}), w)$ is the state reached from start state of M after reading the input string w.

w is accepted by M iff both M_1 and M_2 accept. There exist a a DFSM M that accept $L_1 \cap L_2$. i.e. $L_1 \cap L_2$ is also regular

Scanned by CamScanner

To show that some languages are not Regular :-

* Pumping theorem can be used to show that a language is not regular.
* If the given language fails to satisfy the condition(s) of pumping theorem, the language is not regular.
* According to Pumping theorem, The Regular Language must have following properties or conditions.

1. The Language must contain a long string w such that $|w| \geq n$.
   (Length of string is atleast n)
   where n is some constant.
2. String w must be breakable into 3 substrings x, y & z satisfying the following constraints,
   (i) $|xy| \leq n$ i.e. length of substrings xy is as short as possible.
   (ii) $y \neq \epsilon$ i.e middle string y must not be empty since it is to be deleted or repeated.
   (iii) For all $k \geq 0$, string $xy^k z$ must also be in the language L.

Scanned by CamScanner

State and prove pumping theorem for Regular Languages :-

Statement : Let L be a Regular Language. There exist a Constant n such that for every string w in L such that $|w| \geq n$, the string w is breakable into three substrings x, y & z such that : (i) $y \neq \epsilon$ (ii) $|xy| \leq n$ (iii) For all $k \geq 0$, the string $xy^k z$ is also in L.

That is we can always find a non-empty string y that can be "pumped" that repeating y any number of times or deleting it ($k=0$) keeps the resulting string in the language L.

Proof :- Since L is regular, there exist a DFSM M such that $L(M) = L$. Let the Machine M has n number of states. Consider the string w of length n or more. say $w = a_1 a_2 \ldots a_m$ where $m \geq n$. Each a_i is an input symbol, we can break $w = a_1 a_2 \ldots a_m$ into $w = xyz$ as follows,

(i) $x = a_1 a_2 \ldots a_i$ (ii) $y = a_{i+1} a_{i+2} \ldots a_j$

(iii) $z = a_{j+1} \ldots a_m$.

We need to show that the string $w = a_1 a_2 \ldots a_m$ is accepted by the DFSM.

Scanned by CamScanner

Note that m & n are total symbols of string w and no. of states of FSM respectively & $m \geq n$,

By the Pigeon hole principle, there exist atleast one state that Consumes two or more symbols of the string w.

The following general DFSM that accepts all strings $xy^k z$ where $k = 0, 1, 2, \ldots$.

We are supposed to prove that for all $k \geq 0$, $xy^k z$ is in L. that means show that the DFSM accepts all strings $xy^k z$ for $k \geq 0$ (i.e. $k = 0, 1, 2, \ldots$)

For $k=0$, $xy^2 z = xy^0 z = xz$ must be in L.
From start state q_0 after reading string x the Machine enter state q_i. From q_i after reading string y enters Final state. Therefore xz is accepted, i.e. xz is in L.

For $k > 0$ $xy^k z$ must be in L,
From q_0 on input x FSM enters state q_i on input y^k the Machine moves from q_i to q_i k times. on input z the machine enters accepting state ◎

Therefore DFSM accepts all strings $xy^k z$ for $k \geq 0$.

all

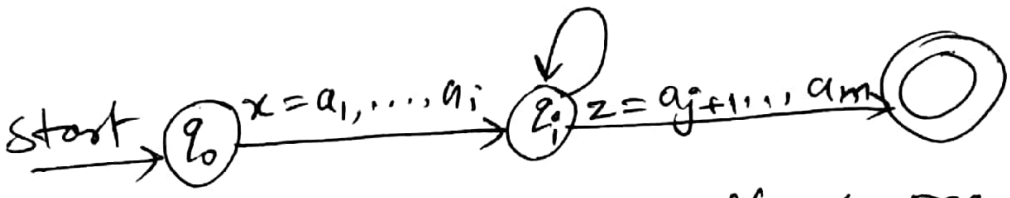


fig:- Long string causes atleast one state to repeat.

Problems: (1) Show that the Language $L = \{a^n b^n : n \geq 1\}$ is not regular.

Proof: $L = \{ab, aabb, aaabbb, \ldots\}$

* Select a long string w such that $|w| \geq n$.
  Let $w = a^n b^n$.
* Break w into substrings x, y & z such that (i) $y \neq \epsilon$ (ii) $|xy| \leq n$.
  Break w into $\underbrace{a^i}_{x}\ \underbrace{a^j}_{y}\ \underbrace{b^n}_{z}$

(i) $y = a^j \neq \epsilon$. (ii) $|xy| = |a^i a^j| = n$.
$\therefore |xy| \leq n$ is satisfied.

Note We broke a^n into $\underbrace{a^i}_{x}$ and $\underbrace{a^j}_{y}$

(iii) for all $k \geq 0$, $xy^k z$ must be in L

$k = 0$ $xy^k z = xy^0 z = xz$ must be in L
$= a^i b^n$ must be in L
$= a^{n-j} b^n$ is not in L

Each string in Language L must be of the form $a^n b^n$. Since j no. of 0's are deleted to produce the string $a^{n-j} b^n$. Therefore L is not Regular Language.

Scanned by CamScanner