FORMAL LANGUAGE THEORY & FINITE AUTOMATA

FINITE AUTOMATA

An abstract machine is called automata
It consist of states and transitions represented by circles & arrows

AUTOMATA : a machine which takes some string as input and this input goes through a finite number of states and may enter in the final state

TERMINOLOGIES USED IN AUTOMATA

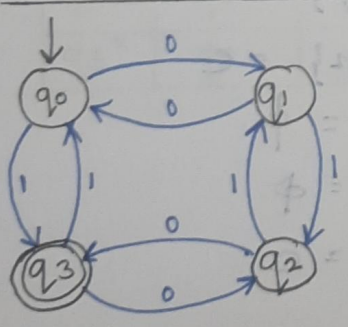
Symbols: are an entity or individual objects, which can be any letter, alphabet or picture. example: 1, a, b, #

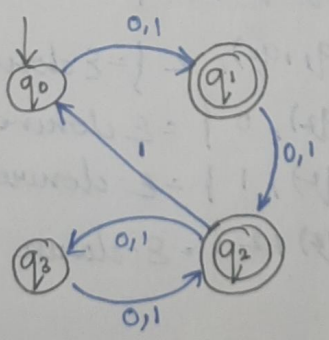
Alphabets: are finite set of symbols, denoted by Σ
example : $\Sigma\{a, b\}$ or $\Sigma\{\varepsilon, \#, \beta\}$

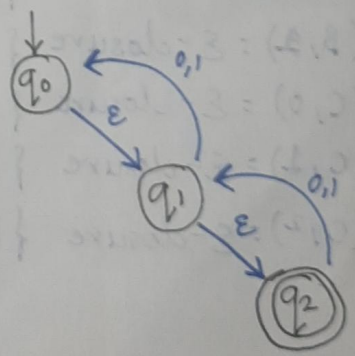
String : finite collection of symbols from the alphabet. denoted by w
If $\Sigma = \{a, b\}$, various strings that can be generated from Σ are $\{a, b, ab, ba, aaa \ldots\}$
A string with zero occurance of symbol is known as an empty string. represented by ε
The number of symbols in string w is called its length and is denoted by $|w|$

Language: is a collection of appropriate strings. A language which is formed over Σ can be finite or infinite.

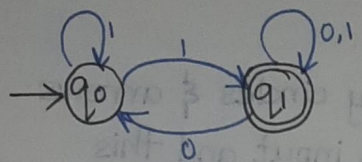
DFA	NDFA	ε - NDFA
Every input has one path	May have many	May have many
Difficult to construct	Easier to construct	Easiest to construct
transition only has the alphabets	alphabets may have multiple transition	transitions are alphabets and ε
has one initial state	has one initial state	can have more than one initial state





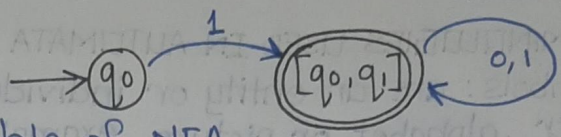


CONVERSION OF NFA TO DFA



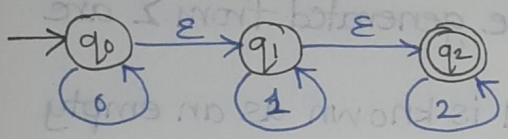
NFA	0	1
q_0	–	$[q_0, q_1]$
q_1	$[q_0, q_1]$	q_1

DFA	0	1
q_0	–	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$



Step 1: write the transition table of NFA.
Step 2: write the transition table of DFA for each new state
Step 3: Draw the FA according to the transition table

CONVERSION OF ε-NFA TO NFA



Step 1: Find ε-closure of all states.

$q_0 = \{q_0, q_1, q_2\}$ $q_1 = \{q_1, q_2\}$ $q_2 = \{q_2\}$

Step 2: Call them as new states.

$\{q_0, q_1, q_2\} = A$ $\{q_1, q_2\} = B$ $\{q_2\} = C$

Step 3:

$\delta'(A,0) = \varepsilon\text{-closure}\{\delta(q_0,q_1,q_2), 0\} = \varepsilon\text{-closure}\{\delta(q_0,0) \cup \delta(q_1,0) \cup \delta(q_2,0)\} = \{q_0 \cup \phi \cup \phi\} = \varepsilon\text{-closure}\{q_0\} = A$

$\delta'(A,1) = \varepsilon\text{-closure}\{\delta(q_0,q_1,q_2), 1\} = \varepsilon\text{-closure}\{q_1\} = B$

$\delta'(A,2) = \varepsilon\text{-closure}\{\delta(q_0,q_1,q_2), 2\} = \varepsilon\text{-closure}\{q_2\} = C$

$\delta'(B,0) = \varepsilon\text{-closure}\{\delta(q_1,q_2), 0\} = \varepsilon\text{ closure}\{\phi\} = \phi$

$\delta'(B,1) = \varepsilon\text{-closure}\{\delta(q_1,q_2), 1\} = \varepsilon\text{ closure}\{q_1\} = B$

$\delta'(B,2) = \varepsilon\text{-closure}\{\delta(q_1,q_2), 2\} = \varepsilon\text{ closure}\{q_2\} = C$

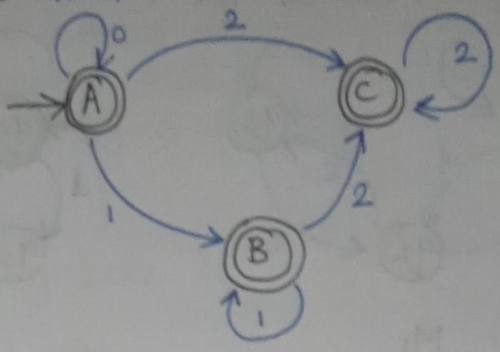
$\delta'(C,0) = \varepsilon\text{-closure}\{\delta(q_2), 0\} = \varepsilon\text{ closure}\{\phi\} = \phi$

$\delta'(C,1) = \varepsilon\text{-closure}\{\delta(q_2), 1\} = \varepsilon\text{-closure}\{\phi\} = \phi$

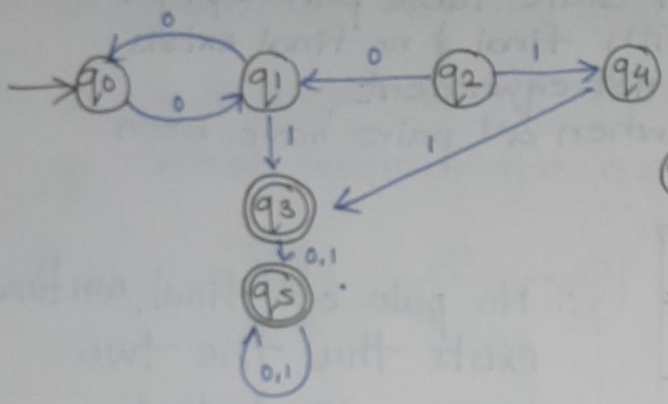
$\delta'(C,2) = \varepsilon\text{-closure}\{\delta(q_2), 2\} = \varepsilon\text{-closure}\{q_2\} = C$

step 4: Draw transition table and FA.

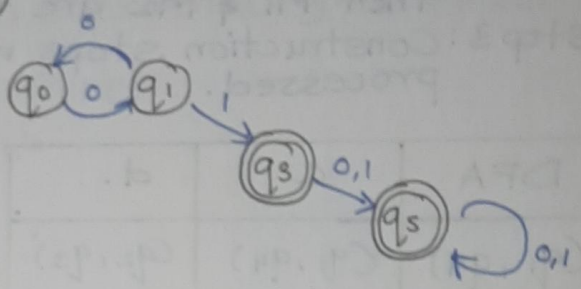
NFA	0	1	2
A	A	B	C
B	ϕ	B	C
C	ϕ	ϕ	C



MINIMIZATION OF DFA



step 1: remove dead states and transitions.



step 2: Draw transition table for remaining ~~tests~~ states

DFA	0	1
q_0	q_1	–
q_1	q_0	q_3

DFA	0	1
q_3^*	q_3	q_3
q_3^*	q_3	q_3

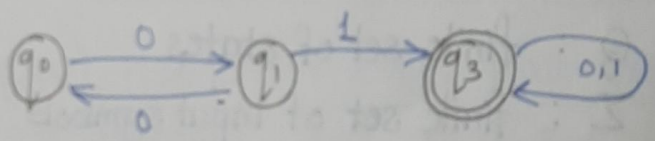
merge

step 3: Seperate final and non final states.

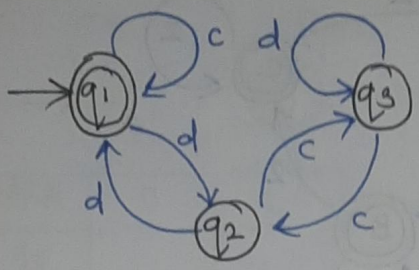
step 4: Merge the states with dublicate outputs.

step 5: Create new state transition table and Draw DFA.

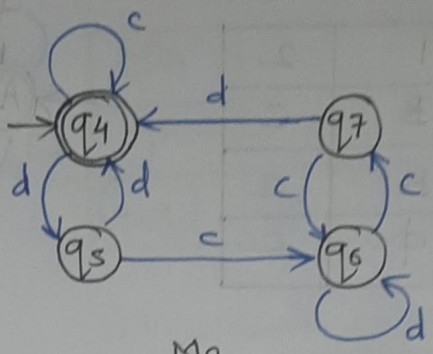
DFA	0	1
q_0	q_1	–
q_1	q_0	q_3
q_3^*	q_3	q_3



EQUIVALENCE OF DFA



M_1



M_2

Step 1: Begin construction of state table pair (p, p')
Step 2: If a output pair with final & no final exists then M_1 & M_2 are not equivalent
Step 3: Construction stops when all pairs have been processed.

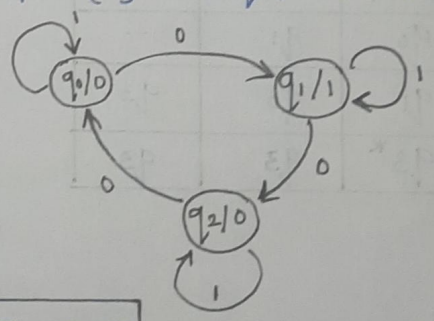
DFA	c	d.
(q_1, q_4)	(q_1, q_4)	$(q_2, q_5)^\circ$
(q_2, q_5)	$(q_3, q_6)^\circ$	(q_1, q_4)
(q_3, q_6)	$(q_2, q_7)^\circ$	(q_3, q_6)
(q_2, q_7)	(q_3, q_6)	(q_1, q_4)

$\therefore$ No pair of (final, nonfinal) exists thus the two DFA are Equivalent.

MOORE MACHINE

Its a Finite state machine in which the next state is decided by the current state and current input ~~signal~~ symbol
The output symbol at a given time depends only on the present state of the machine
Moore machine can be described by 6 tuple $(Q, \Sigma, q_0, O, \delta, A)$

Q : finite set of states
Σ : finite set of input symbols
q_0 : initial state of machine
O : output alphabet
δ : transition function where $Q \times \Sigma \to Q$
λ : output function where $Q \to O$



MEALY MACHINE

Its a machine in which output symbol depends upon the present input symbol and present state of the machine

Mealy machine can be described by 6 tuples

Q : finite set of states

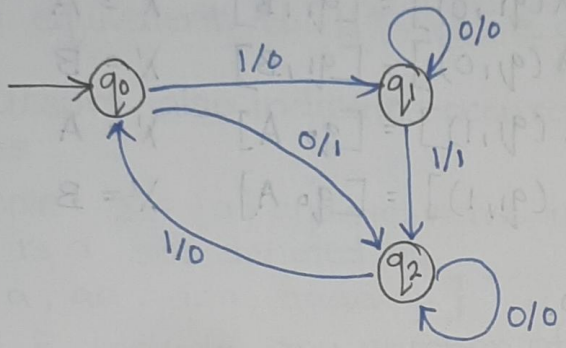
Σ : is finite set of input symbols

Δ : is an output alphabet set

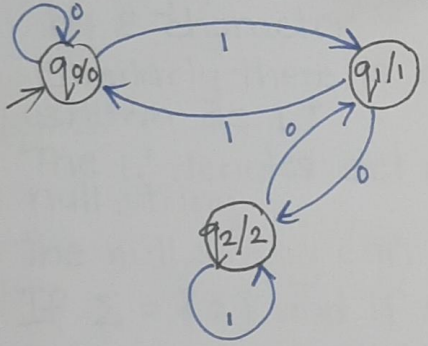
δ : transition function such that $Q \times \Sigma \to Q$

λ : is a machine function such that $Q \times \Sigma \to \Delta$

q_0 : initial state of the machine.



CONVERSION OF MOORE TO MEALY MACHINE



Moore	0	1	O/P
q_0	q_0	q_1	0
q_1	q_2	q_0	1
q_2	q_1	q_2	2

Output fxn : $\lambda'(q,a) = \lambda(\delta(q,a))$, Hence for every transition

$\lambda'(q_0,0) = \lambda(\delta(q_0,0)) = \lambda(q_0) = 0$

$\lambda'(q_0,1) = \lambda(\delta(q_0,1)) = \lambda(q_1) = 1$

$\lambda'(q_1,0)\ \ \lambda(\delta(q_1,0)) = \lambda(q_2) = 2$

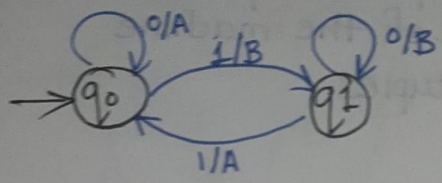
$\lambda'(q_1,1)\ \ \lambda(\delta(q_1,1)) = \lambda(q_0) = 0$

$\lambda'(q_2,0)\ \ \lambda(\delta(q_2,0)) = \lambda(q_1) = 1$

$\lambda'(q_2,1)\ \ \lambda(\delta(q_2,1)) = \lambda(q_2) = 2$

Mealy	0	O/P	1	O/P
q_0	q_0	0	q_1	1
q_1	q_2	2	q_0	0
q_2	q_1	1	q_2	2

CONVERSION OF MEALY TO MOORE MACHINE



Step 1 Calculate δ' & λ'

$\delta'((q_0, A), 0) = [\delta(q_0, 0), \lambda(q_0, 0)] = [q_0, A]$ $\quad \lambda' = A$

$\delta'((q_0, B), 0) = [\delta(q_0, 0), \lambda(q_0, 0)] = [q_0, A]$ $\quad \lambda' = B$

$\delta'((q_0, A), 1) = [\delta(q_0, 1), \lambda(q_0, 1)] = [q_1, B]$ $\quad \lambda' = A$

$\delta'((q_0, B), 1) = [\delta(q_0, 1), \lambda(q_0, 1)] = [q_1, B]$ $\quad \lambda' = B$

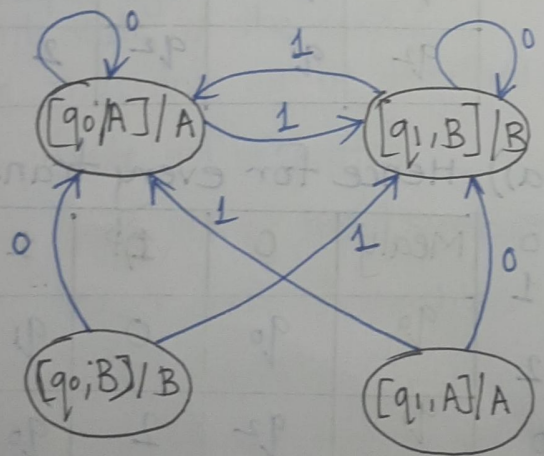
$\delta'((q_1, A), 0) = [\delta(q_1, 0), \lambda(q_1, 0)] = [q_1, B]$ $\quad \lambda' = A$

$\delta'((q_1, B), 0) = [\delta(q_1, 0), \lambda(q_1, 0)] = [q_1, B]$ $\quad \lambda' = B$

$\delta'((q_1, A), 1) = [\delta(q_1, 1), \lambda(q_1, 1)] = [q_0, A]$ $\quad \lambda' = A$

$\delta'((q_1, B), 1) = [\delta(q_1, 1), \lambda(q_1, 1)] = [q_0, A]$ $\quad \lambda' = B$

MOORE	0	1	o/p.
$[q_0, A]$	$[q_0, A]$	$[q_1, B]$	A
$[q_0, B]$	$[q_0, A]$	$[q_1, B]$	B
$[q_1, A]$	$[q_1, B]$	$[q_0, A]$	A
$[q_1, B]$	$[q_1, B]$	$[q_0, A]$	B



UNIT 2

REGULAR EXPRESSION

Let Σ be an alphabet which is used to denote the input set
The regular expression over Σ can be defined as follows –

1. ϕ is a regular expression which denotes the empty set
2. ε is a regular expression and denotes the set $\{\varepsilon\}$ and it is a null string
3. For each 'a' in Σ, 'a' is a regular expression denoting the languages L_1 & L_2 respectively then.
4. If r and s are regular expressions denoting the languages L_1 and L_2 respectively, then

$r+s$ is equivalent to $L_1 \cup L_2$ i.e union.
rs is equivalent to $L_1 L_2$ i.e concatenation
r^* is equivalent to L^* i.e closure.
also known as kleen closure

Kleen closure also indicates occurance of r for ∞ number of times
for example : $\Sigma = \{a\}$ and we have regular expression $R = a^*$, then R is a set denoted by
$R = \{\varepsilon, a, aa, aaa, aaaa \ldots\}$
That is R include any number of 'a's as well as empty string which indicates zero number of a's appearing denoted by ε character
Similarly there is a positive closure of L which can be shown as L^+
The L^+ denotes set of all the string except the ε or null string.
The null string can be represented by ε or $\wedge$
If $\Sigma = \{a\}$ and if we have regular expression $R = a^+$ then, R is a set denoted by $R = \{a, aa, aaa, aaaa \ldots\}$
We can construct L^* as $L^* = \varepsilon \cdot L^+$

IDENTITIES OF REGULAR EXPRESSION AND EQUIVALENCE OF TWO REGULAR EXPRESSION

The two regular expression P & Q are equivalent (denoted as P=Q) iff P represents the same set of strings as Q does.

For showing this equivalence of regular expression we need to show some identities of regular expression

Let P, Q and R be regular expressions then the identity rules are given below.

1. $\varepsilon R = R\varepsilon = R$
2. $\varepsilon^* = \varepsilon$ } null strings
3. $(\phi)^* = \varepsilon$ }
4. $\phi R = R\phi = \phi$
5. $\phi + R = R + \phi = R$
6. $R + R = R$
7. $RR^* = R^*R = R^+$
8. $(R^*)^* = R^*$
9. $\varepsilon + RR^* = R^*$
10. $(P+Q)R = PR + QR$
11. $(P+Q)^* = (P^*Q^*) = (P^* + Q^*)^*$
12. $R^*(\varepsilon + R) = (\varepsilon + R)R^* = R^*$
13. $(R+\varepsilon)^* = R^*$
14. $\varepsilon + R^* = R^*$
15. $(PQ)^*P = P(QP)^*$
16. $R^*R + R = R^*R$

SHOW THAT $(0^*1^*)^* = (0+1)^*$

LHS $= (0^*1^*)^* = \{\varepsilon, 0, 00, 000\ldots, 1, 11, 111\ldots, 01, 10\ldots\}$

RHS $= (0+1)^* = \{\varepsilon, 0, 00, 000, \ldots, 1, 11, 111\ldots, 01, 10\ldots\}$

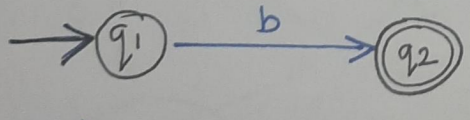
This means both regular expression allows any combination of 0's and 1's

CONSTRUCT NFA FOR THE REGULAR EXPRESSION $b + ba^*$

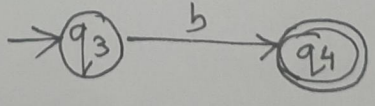
The regular expression can be broken in r_1 and r_2 as

$r_1 = b$ & $r_2 = ba^*$

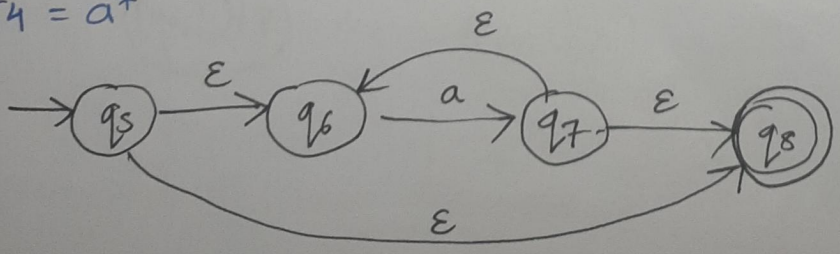
Lets draw NFA for r_1, which is very simple



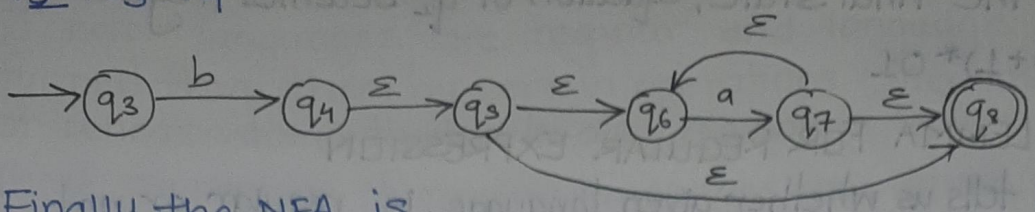
r_2 { $r_3 = b$ $r_4 = a^*$ }



$r_4 = a^*$

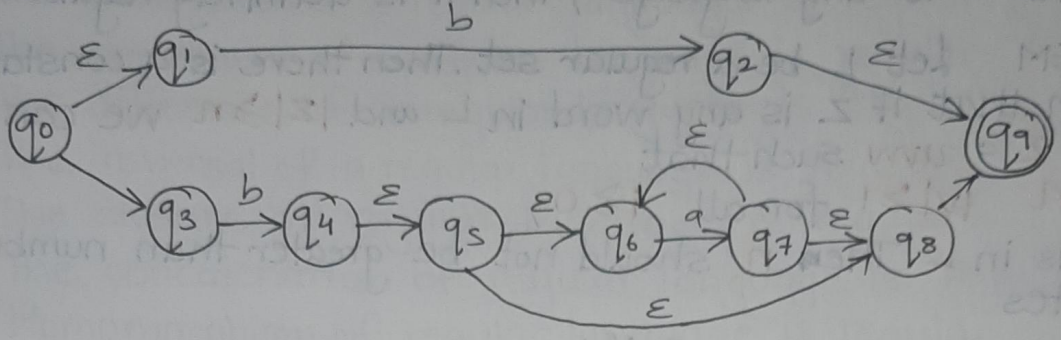


$r_2 = r_3 \cdot r_4$



Finally the NFA is.

$r_0 = r_1 + r_2$

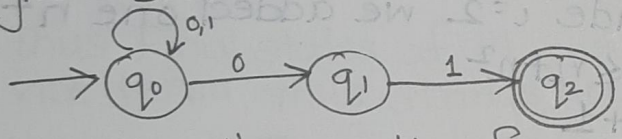


CONVERSION OF FA TO RE USING ARDEN'S THEOREM

Let P and Q be then true regular expression over the input set Σ

The regular expression R is given as $R = Q + RP$ which has unique solution as $R = QP^*$

Example : Construct RE by using Ardens Theorem for given DFA.



We will write equation for each state.

$q_0 = q_0 0 + q_0 1 + \varepsilon$ — (1)

$q_1 = q_0 0$ — (2)

$q_2 = q_1 1$ — (3)

$q_0 = q_0(0+1) + \varepsilon$

$R = R(P) Q$

$R = RP + Q$

$R = QP^*$

By Ardens theorem, if $R = QP^*$

$\therefore q_0 = \varepsilon(0+1)^*$ $\therefore q_0 = (0+1)^*$ — (4)

$q_1 = q_0 0$ $\therefore q_1 = (0+1)^* 0$ — (5)

$q_2 = q_1 1$ putting (5) in (3) we get $q_2 = (0+1)^* 01$

As q_2 is the final state, equation of q_2 becomes regular

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

PUMPING LEMMA FOR REGULAR EXPRESSION

This lemma tells us whether given language is regular or not one key idea is that ; if it is possible to design the finite automata for any language ; Then it is definitely regular

THEOREM Let L be a regular set. Then there is a constant n such that if z is any word in L and $|z| \geq n$ we can write $z = uvw$ such that

$|uv| \leq n$ $|v| \geq 1$ for all $i \geq 0$,

uv^iw is in L Then n should not be greater than number of states

Example: Show set $L = \{a^{i^2} | i > 1\}$ is not regular.

This language is such that the number of a's are always perfect square $L = a^{1^2} = a. = a^{2^2} = aaaa$; length $= 2^2$

$L = a^{n^2}$ where length n^2 it is denoted by z

$|z| = n^2$

By pumping lemma $z = uvw$ where $1 \leq |v| \leq n$

as $z = uv^iw$ where $i = 1$

now we pump vu make $i = 1$

As we made $i=2$ As we made $i=2$ we added one n^2

$1 \leq |v| \leq n$. $n^2+1 \leq |uvw| \leq n+n^2$

i.e $n^2+1 \leq |uvw| \leq n^2+n+n+1$

i.e $n^2+1 \leq |uvw| \leq (n+1)^2$

$= n^2 \leq |uvw| \leq (n+1)^2$

Thus the string lies between consequence pen consequtive perfect squares.

Hence given language is not regular

CLOSURE PROPERTY OF REGULAR EXPRESSION

If certain languages are regular and language L is formed from them by certain operations (such as union or concatenation) then L is also regular

These properties of regular languages are given below

1. The union of two regular languages is regular.
2. The intersection of 2 regular languages is regular.
3. The complement of a regular language is regular.
4. The difference of 2 regular language is regular.
5. The reversal of a regular language is regular.
6. The closure of regular language is regular.
7. The concatenation of regular language is regular.
8. Homomorphism of regular language is regular.
9. Inverse Homomorphism of regular language is regular.

✱ THEOREM I : If L_1 & L_2 are two language then $L_1 \cup L_2$ is regular.

✱ THEOREM II : The complement of regular language is regular.

Consider L_1 accepted by $M = (Q, \Sigma, \delta, q_0, F)$ ∴

∴ Complement L_1 i.e $M' = (Q, \Sigma, \delta, q_0, Q-F)$

Thus Final states of M are non-final of M' and vice versa thus strings accepted by M are rejected by M'

✱ THEOREM III : The intersection of L_1 & L_2 i.e $L_1 \cap L_2$ is regular.

L_1 accepted by $M_1 = (Q_1, \Sigma_1, \delta_1, q_0, F_1)$

L_2 accepted by $M_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, F_2)$

$\therefore Q = Q_1 \cap Q_2$, $\delta = \delta_1 \cap \delta_2$, $q \in Q$ & $F = F_1 \cap F_2$

for a machine M where $M_1 \cap M_2$

thus L accepted by M is $L_1 \cap L_2$ which is regular

REVERSAL OF DFA

Steps:

1. Draw states as it is
2. Make final states as initial and vice versa.
3. Reverse the edges
4. Loops will remain same
5. Remove inappropriate transition states.

NOTE Not all reversal of DFA leads to DFA.
reversal of DFA can create a NFA

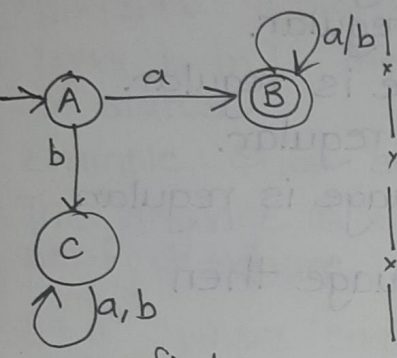


fig 1.

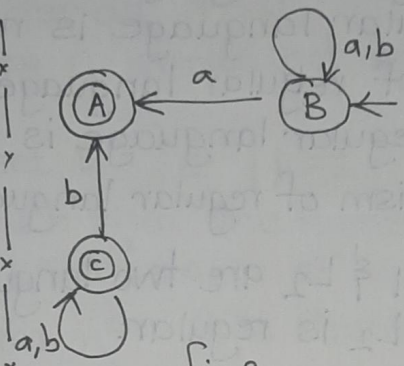


fig 2

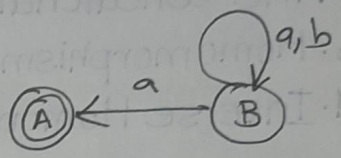
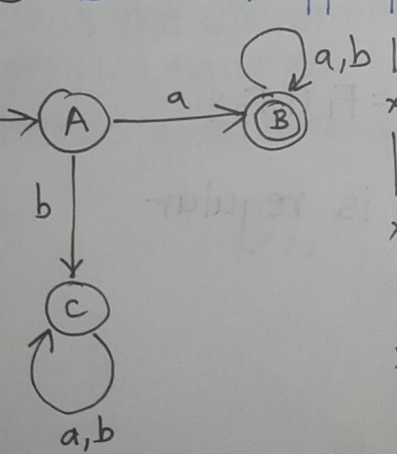


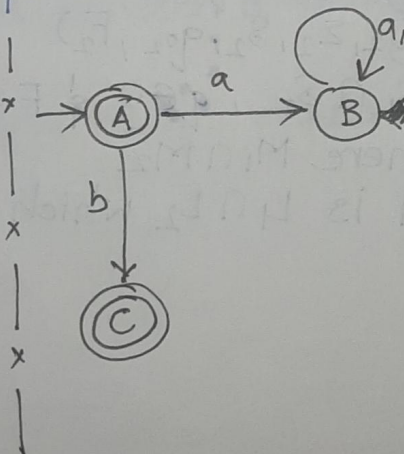
fig 3

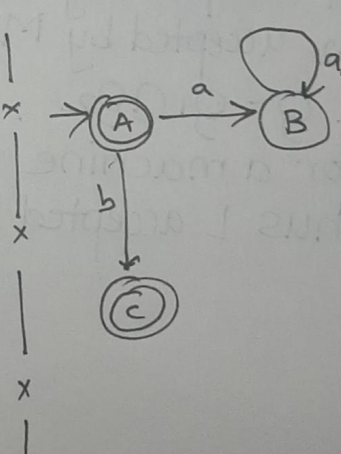
COMPLEMENT OF DFA.

Steps:

1. Draw states as it is
2. Make final state as initial & vice versa.
3. Loops will remain same
4. Edges will remain same
5. Remove inappropriate transition states.





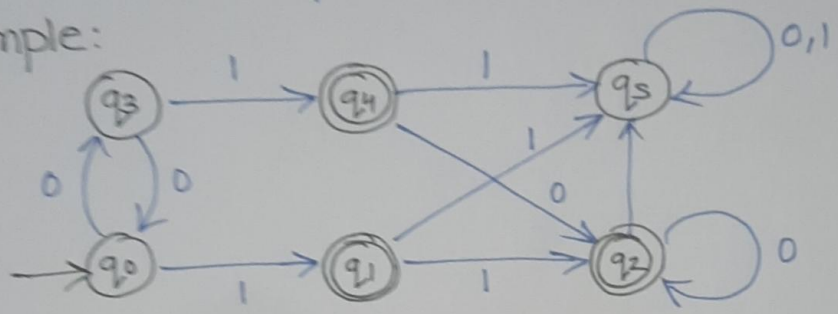


Myhill-Nerode THEOREM (minimization of DFA)

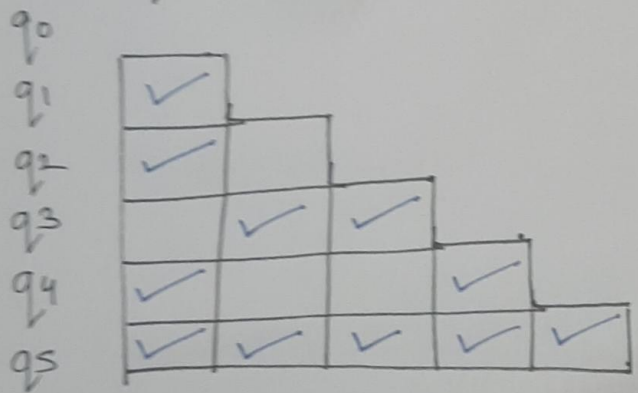
Steps

1. Create a Matrix
   Cutshort the upper diagonal matrix
2. Mark the cells with final, non final pair
3. Give value 0, 1 to unmarked cells and watch the transition
   If the o/p pair is marked: mark the cell
4. Iterate over step 3 until all pair are marked.
5. Consider unmarked pair as seperate states
6. If final, final pair then state is final.

Example:



q_0 q_1 q_2 q_3 q_4 q_5



- unmarked pairs.

$[q_1, q_2]$ $[q_0, q_3]$
$[q_1, q_4]$ $[q_2, q_4]$

- After taking common

$[q_1, q_2, q_4]$ $[q_0, q_3]$

Final DFA.