UNIT 2

Syllabus: Regular Expressions, Finite Automata and Regular Expressions, Applications of Regular Expressions, Algebraic Laws for Regular Expressions, Properties of Regular Languages- Pumping Lemma for Regular Languages, Applications of the Pumping Lemma, Closure Properties of Regular Languages, Decision Properties of Regular Languages, Equivalence and Minimization of Automata.

==

Regular Expression can be recursively defined as follows:

1. ε is a Regular Expression indicates the language containing an empty string. (L (ε) = {ε})
2. φ is a Regular Expression denoting an empty language. (L (φ) = { })
3. x is a Regular Expression where L={x}
4. If X is a Regular Expression denoting the language L(X) and Y is a Regular Expression denoting the language L(Y), then

   (a) X + Y is a Regular Expression corresponding to the language L(X) U L(Y) where L(X+Y) = L(X) U L(Y).
   X . Y is a Regular Expression corresponding to the language L(X) . L(Y) where L(X.Y)= L(X) . L(Y)

   (b) R* is a Regular Expression corresponding to the language L(R*) where L(R*) = (L(R))*
5. If we apply any of the rules several times from 1 to 5, they are Regular Expressions.

Some RE Examples

Regular Expression	Regular Set
(0+10*)	L= { 0, 1, 10, 100, 1000, 10000, … }

(0*10*)	L={1, 01, 10, 010, 0010, …}
(0+ε)(1+ ε)	L= {ε, 0, 1, 01}
(a+b)*	Set of strings of a's and b's of any length including the null string. So L= { ε, a, b, aa , ab , bb , ba, aaa…….}
(a+b)*abb	Set of strings of a's and b's ending with the string abb. So L = {abb, aabb, babb, aaabb, ababb, …………..}
(11)*	Set consisting of even number of 1's including empty string,
	So L= {ε, 11, 1111, 111111, ……….}
(aa)*(bb)*b	Set of strings consisting of even number of a's followed by odd number of b's , so L= {b, aab, aabbb, aabbbbb, aaaab, aaaabbb, …………..}
(aa + ab + ba + bb)*	String of a's and b's of even length can be obtained by concatenating any combination of the strings aa, ab, ba and bb including null, so L= {aa, ab, ba, bb, aaab, aaba, …………..}

Regular Sets

Any set that represents the value of the Regular Expression is called a Regular Set.

Properties of Regular Sets/ Regular Languages

Property 1 *The union of two regular set is regular.*

Proof*:*

Let us take two regular expressions

RE1 = a(aa)* and RE2 = (aa)*

So, L1= {a, aaa, aaaaa,.....} (Strings of odd length excluding Null) and

L2={ ε, aa, aaaa, aaaaaa,.......} (Strings of even length including Null) L1 ∪ L2 = { ε, a, aa, aaa, aaaa, aaaaa, aaaaaa,.......}

(Strings of all possible lengths including Null)

RE (L1 ∪ L2) = a* (which is a regular expression itself)

Hence, proved.

Property 2 *The intersection of two regular set is regular.*

Proof*:*

Let us take two regular expressions

RE1 = a(a*) and RE2 = (aa)* So,

L1 = { a,aa, aaa, aaaa, ...} (Strings of all possible lengths excluding Null) L2 ={ ε, aa, aaaa, aaaaaa,.......} (Strings of even length including Null)

L1 ∩ L2 = { aa, aaaa, aaaaaa,.......} (Strings of even length excluding Null)

RE (L1 ∩ L2) = aa(aa)* which is a regular expression itself.

Hence, proved.

Property 3 *The complement of a regular set is regular.*

Proof*:* Let us take a regular expression: RE = (aa)*

So, $L = \{\varepsilon, aa, aaaa, aaaaaa, ...\}$ (Strings of even length including Null)

Complement of L is all the strings that is not in L.

So, $L' = \{a, aaa, aaaaa, ...\}$ (Strings of odd length excluding Null)

RE (L') = a(aa)* which is a regular expression itself.

Hence, proved.

Property 4 *The difference of two regular set is regular.*

Proof:

Let us take two regular expressions:

RE1 = a (a*) and RE2 = (aa)* So,

L1= {a, aa, aaa, aaaa, ………….} (Strings of all possible lengths excluding Null)

L2 = { ε, aa, aaaa, aaaaaa,.......} (Strings of even length including Null)

L1 – L2 = {a, aaa, aaaaa, aaaaaaa, ...} (Strings of all odd lengths excluding Null)

RE (L1 – L2) = a (aa)* which is a regular expression.

Hence, proved.

Property 5 *The reversal of a regular set is regular.*

Proof:

We have to prove L^R is also regular if L is a regular set. Let,

L= {01, 10, 11}

RE (L)= 01 + 10 + 11

L^R= {10, 01, 11}

RE (L^R)= 10+ 01+ 11 which is regular

Hence, proved.

Property 6 *The closure of a regular set is regular.*

Proof*:*

If L = {a, aaa, aaaaa, ...} (Strings of odd length excluding Null) i.e., RE (L) = a (aa)*

L*= {a, aa, aaa, aaaa , aaaaa,……………} (Strings of all lengths excluding Null)

RE (L*) = a (a)*

Hence, proved.

Property 7 *The concatenation of two regular sets is regular.*

Proof*:*

Let RE1 = (0+1)*0 and RE2 = 01(0+1)*

Here, L1 = {0, 00, 10, 000, 010, ...} (Set of strings ending in 0)

L2 = {01, 010,011,.....} (Set of strings beginning with 01) Then,

L1 L2 = {010, 0100, 0110, 0110, 011110, 011000, 010010,.............}

Set of strings containing 010 as a substring which can be represented by an RE: 01(0+1)*0

Hence, proved.

Identities Related to Regular Expressions

Given R, P, L, Q as regular expressions, the following identities hold:

1. Ø* = ε

2. ε* = ε

3. RR* = R*R

4. 4. R*R* = R*

5. 5. (R*)* = R*

6. RR* = R*R

7. $(PQ)^*P = P(QP)^*$

8. $(a+b)^* = (a^*b^*)^* = (a^*+b^*)^* = (a+b^*)^* = a^*(ba^*)^*$

9. $R + Ø = Ø + R = R$ (The identity for union)

10. $R\varepsilon = \varepsilon R = R$ (The identity for concatenation)

11. $ØL = LØ = Ø$ (The annihilator for concatenation)

12. $R + R = R$ (Idempotent law)

13. $L (M + N) = LM + LN$ (Left distributive law)

14. $(M + N) L = LM + LN$ (Right distributive law)

15. $\varepsilon + RR^* = \varepsilon + R^*R = R^*$

Arden's Theorem

In order to find out a regular expression of a Finite Automaton, we use Arden's Theorem along with the properties of regular expressions.

Statement*:*

Let P and Q be two regular expressions.

If P does not contain null string, then $R = Q + RP$ has a unique solution that is $R = QP^*$

Proof*:*

$R = Q + (Q + RP)P$ [After putting the value $R = Q + RP$]

$= Q + QP + RPP$

When we put the value of **R** recursively again and again, we get the following equation:

$R = Q + QP + QP^2 + QP^3$..... $R = Q (\epsilon + P + P^2 + P^3 + ...)$

$R = QP^*$[As P^* represents $(\epsilon + P + P^2 + P^3 + ...)$] Hence, proved.

Assumptions for Applying Arden's Theorem

1. The transition diagram must not have NULL transitions
2. It must have only one initial state

Method

Step 1: Create equations as the following form for all the states of the DFA having

n states with initial state q1.

$q1 = q1R11 + q2R21 + … + qnRn1 + \epsilon$ $q2 = q1R12 + q2R22 + … + qnRn2$

....

....

....

....

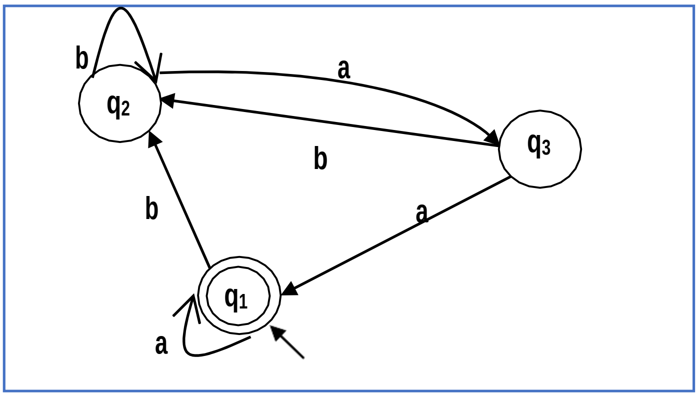
$qn = q1R1n + q2R2n + … + qnRnn$

Rij represents the set of labels of edges from **qi** to **qj**, if no such edge exists, then **Rij = Ø**

Step 2: Solve these equations to get the equation for the final state in terms of **Rij**

Problem

Construct a regular expression corresponding to the automata given below:



Finite automata

Solution

Here the initial state is **q2** and the final state is **q1**.

The equations for the three states q1, q2, and q3 are as follows:

q1 = q1a + q3a + ϵ (ϵ move is because q1 is the initial state0 q2 = q1b + q2b + q3b

q3 = q2a

Now, we will solve these three equations:

q2 = q1b + q2b + q3b

= q1b + q2b + (q2a)b (Substituting value of q3)

= q1b + q2(b + ab)

= q1b (b + ab)* (Applying Arden’s Theorem)

q1 = q1a + q3a + ϵ

= q1a + q2aa + ϵ (Substituting value of q3)

= q1a + q1b(b + ab*)aa + ϵ (Substituting value of q2)

= q1(a + b(b + ab)*aa) + ϵ

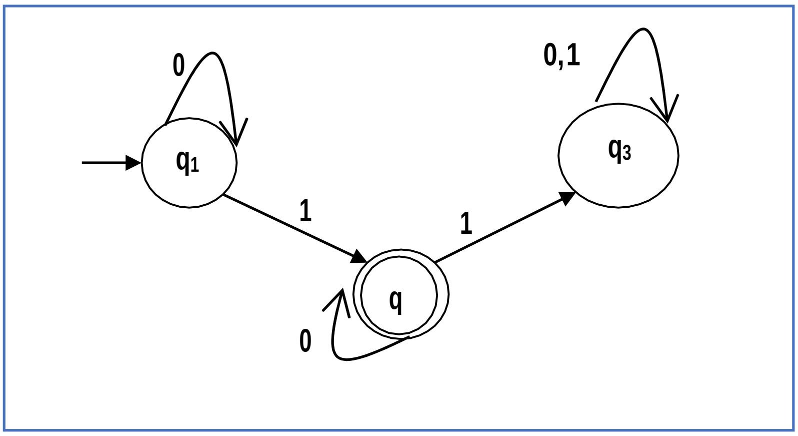
= ϵ (a+ b(b + ab)*aa)*

= (a + b(b + ab)*aa)*

Hence, the regular expression is (a + b(b + ab)*aa)*.

Problem

Construct a regular expression corresponding to the automata given below:



Finite automata

Solution Here the initial state is q1 and the final state is q2 Now we write down the equations:

q1 = q10 + ϵ q2 = q11 + q20

q3 = q21 + q30 + q31

Now, we will solve these three equations: q1 = ϵ0* [As, εR = R]

So, q1 = 0*

q2 = 0*1 + q20

So, q2 = 0*1(0)* [By Arden’s theorem]

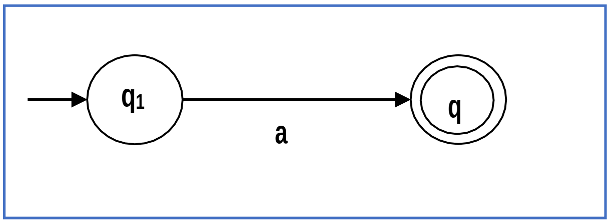
Hence, the regular expression is 0*10*.

Construction of a FA from an RE

We can use Thompson's Construction to find out a Finite Automaton from a Regular Expression. We will reduce the regular expression into smallest regular expressions and converting these to NFA and finally to DFA.

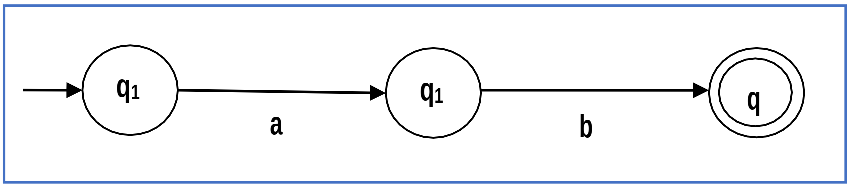
Some basic RA expressions are the following:

Case 1: For a regular expression 'a', we can construct the following FA:



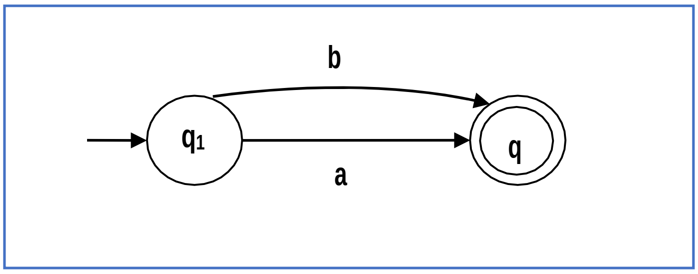
Finite automata for RE = a

Case 2: For a regular expression 'ab', we can construct the following FA:



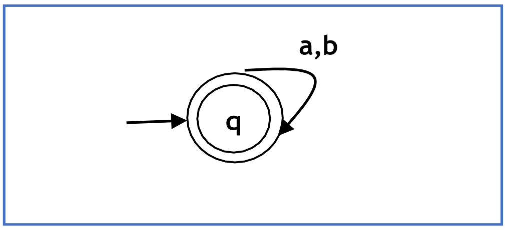
Finite automata for RE = ab

Case 3: For a regular expression (a+b), we can construct the following FA:



Finite automata for RE= (a+b)

***Case 4*:** For a regular expression (a+b)*, we can construct the following FA:



Finite automata for RE= (a+b)*

Method:

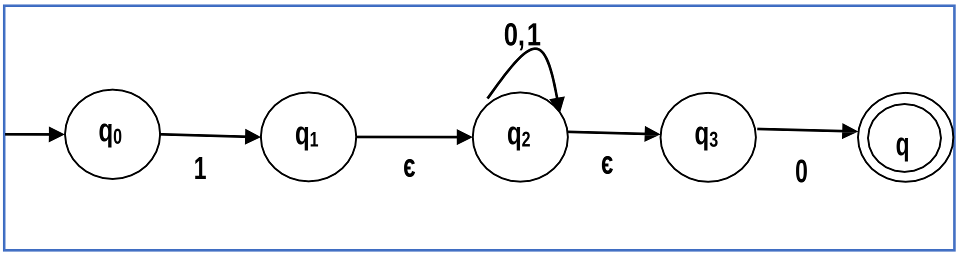
Step 1 Construct an NFA with Null moves from the given regular expression.

Step 2 Remove Null transition from the NFA and convert it into its equivalent DFA.

Problem Convert the following RA into its equivalent DFA: 1 (0 + 1)* 0

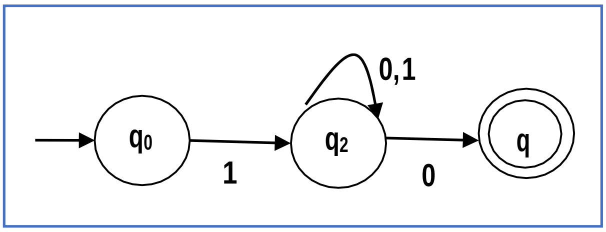
Solution*:*

We will concatenate three expressions "1", "(0 + 1)*" and "0"



NDFA with NULL transition for RA: 1 (0 + 1)* 0

Now we will remove the **ϵ** transitions. After we remove the **ϵ** transitions from the NDFA, we get the following:



NDFA without NULL transition for RA: 1 (0 + 1)* 0

It is an NDFA corresponding to the RE: 1 (0 + 1)* 0. If you want to convert it into a DFA, simply apply the method of converting NDFA to DFA discussed in Chapter 1.

Properties of Regular Languages The Pumping Lemma for Regular Languages, Applications of the Pumping Lemma Closure Properties of Regular Languages, Decision Properties of Regular Languages, Equivalence and Minimization ofAutomata,

Context-Free Grammars and Languages: Definition of Context-Free Grammars, Derivations Using a Grammars Leftmost and Rightmost Derivations, The Languages of a Grammar,

Parse Trees: Constructing Parse Trees, The Yield of a Parse Tree, Inference Derivations, and Parse Trees, From Inferences to Trees, From Trees to Derivations, From Derivation to Recursive Inferences,

Applications of Context-Free Grammars: Parsers, Ambiguity in Grammars and Languages: Ambiguous Grammars, Removing Ambiguity From Grammars, Leftmost Derivations as a Way to Express Ambiguity, InherentAnbiguity.

Pumping Lemma for Regular Languages

Theorem

Let **L** be a regular language. Then there exists a constant '**c**' such that for every string

w in **L:**

$|w| \geq c$

We can break **w** into three strings, **w = xyz**, such that:

1. $|y| > 0$
2. $|xy| \leq c$
3. For all $k \geq 0$, the string xy^kz is also in L.

Applications of Pumping Lemma

Pumping Lemma is to be applied to show that certain languages are not regular. It should never be used to show a language is regular.

1. If **L** is regular, it satisfies Pumping Lemma.
2. If **L** does not satisfy Pumping Lemma, it is non-regular.

Method to prove that a language L is not regular:

1. At first, we have to assume that **L** is regular.
2. So, the pumping lemma should hold for **L**.
3. Use the pumping lemma to obtain a contradiction:

a. Select w such that $|w| \geq c$

b. Select y such that $|y| \geq 1$

c. Select x such that $|xy| \leq c$

d. Assign the remaining string to **z**.

e. Select k such that the resulting string is not in L. L is not regular.

Problem

Prove that **L = {$a^i b^i$ | i ≥ 0}** is not regular.

Solution*:*

1. At first, we assume that **L** is regular and **n** is the number of states.
2. Let w = $a^n b^n$. Thus |w| = 2n ≥ n.
3. By pumping lemma, let w = xyz, where |xy|≤ n.
4. Let x = a^p, y = a^q, and z = $a^r b^n$, where p + q + r = n, p ≠ 0, q ≠ 0, r ≠ 0. Thus |y|≠ 0.
5. Let k = 2. Then $xy^2z = a^p a^{2q} a^r b^n$.
6. Number of ***a***s = (p + 2q + r) = (p + q + r) + q = n + q
7. Hence, $xy^2z = a^{n+q} b^n$. Since q ≠ 0, xy^2z is not of the form $a^n b^n$.
8. Thus, xy^2z is not in L. Hence L is not regular.

Equivalence of Regular Expressions and NFA-ε

• **Note:** Throughout the following, keep in mind that a string is accepted by an NFA-ε if there exists a path from the start state to a final state.

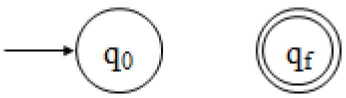
• **Lemma 1:** Let r be a regular expression. Then there exists an NFA-ε M such that L(M) = L(r). Furthermore, M has exactly one final state with no transitions out of it.

• **Proof:** (by induction on the number of operators, denoted by OP(r), in r).

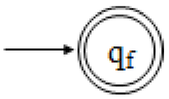
• **Basis:** OP(r) = 0

Then r is either Ø, ε, or **a**, for some symbol **a** in Σ

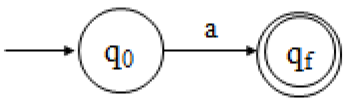
For Ø:



For ε:



For **a**:



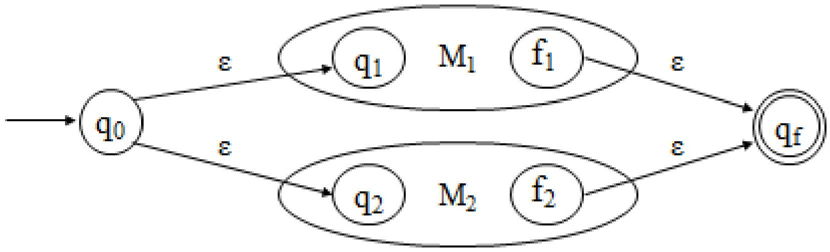
Inductive Hypothesis: Suppose there exists a k ☐ 0 such that for any regular expression r where 0 ☐ OP(r) ☐ k, there exists an NFA-ε such that L(M) = L(r). Furthermore, suppose that M has exactly one final state.

• **Inductive Step:** Let r be a regular expression with k + 1 operators (OP(r) = k + 1), where k + 1 >= 1.

Case 1) r = r1 + r2

Since OP(r) = k +1, it follows that 0<= OP(r1), OP(r2) <= k. By the inductive hypothesis there exist NFA-ε machines M1 and M2 such that L(M1) = L(r1) and L(M2) = L(r2). Furthermore, both M1 and M2 have exactly one final state.

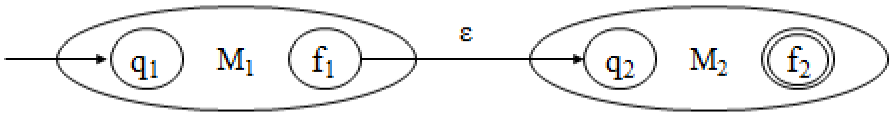
Construct M as:



Case 2) r = r1r2

Since OP(r) = k+1, it follows that 0<= OP(r1), OP(r2) <= k. By the inductive hypothesis there exist NFA-ε machines M1 and M2 such that L(M1) = L(r1) and L(M2) = L(r2). Furthermore, both M1 and M2 have exactly one final state.

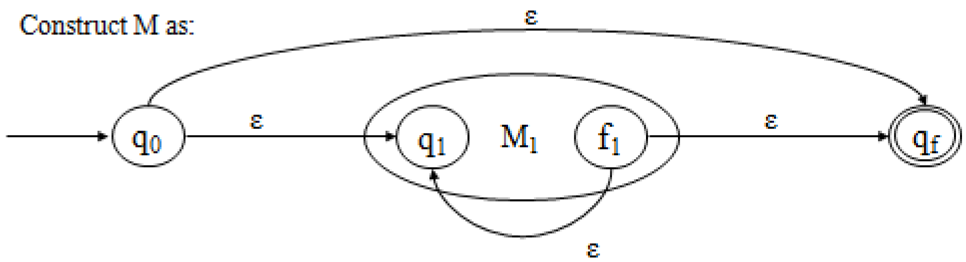
Construct M as:



Case 3) r = r1*

Since OP(r) = k+1, it follows that 0<= OP(r1) <= k. By the inductive hypothesis there exists an NFA-ε machine M1 such that L(M1) = L(r1). Furthermore, M1 has exactly one final state.

Construct M as:



• **Example:**

Problem: Construct FA equivalent to RE, r = 0(0+1)*

Solution: r = r1r2

r1 = 0

r2 = (0+1)*

r2 = r3*

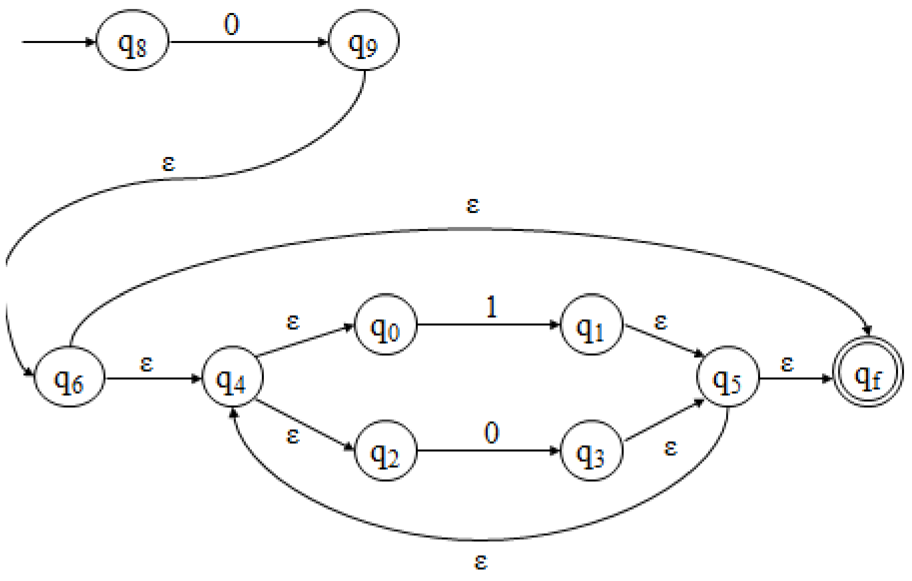
r3 = 0+1

r3 = r4 + r5

r4 = 0

r5 = 1

Transition graph:



DFA Minimization using Equivalence Theorem

If X and Y are two states in a DFA, we can combine these two states into {X, Y} if they are not distinguishable. Two states are distinguishable, if there is at least one string S, such that one of δ (X, S) and δ (Y, S) is accepting and another is not accepting. Hence, a DFA is minimal if and only if all the states are distinguishable.

Algorithm 3

Step 1: All the states **Q** are divided in two partitions: **final states** and **non-final states** and are denoted by **P0**. All the states in a partition are 0^{th} equivalent. Take a counter **k** and initialize it with 0.

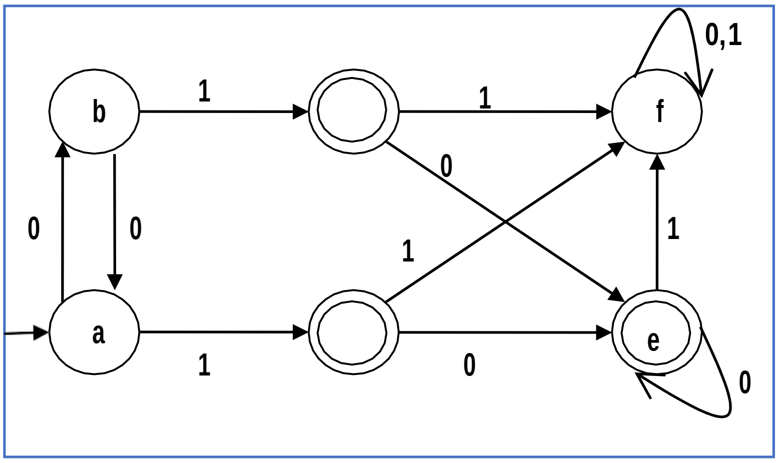
Step 2: Increment k by 1. For each partition in Pk, divide the states in Pk into two partitions if they are k-distinguishable. Two states within this partition X and Y are k-distinguishable if there is an input **S** such that **δ(X, S)** and **δ(Y, S)** are (k-1)-distinguishable.

Step 3: If Pk ≠ Pk-1, repeat Step 2, otherwise go to Step 4.

Step 4: Combine kth equivalent sets and make them the new states of the reduced DFA.

Example

Let us consider the following DFA:



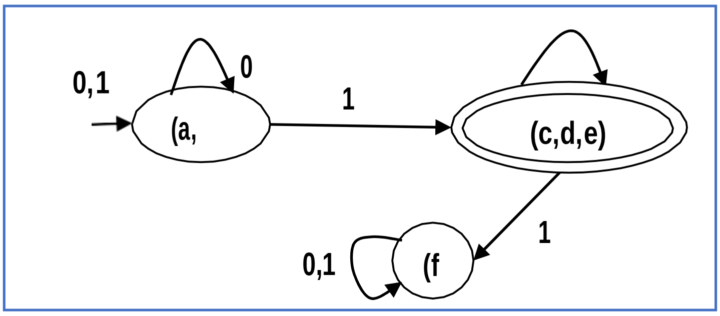
q	δ(q,0)	δ(q,1)
a	b	c
b	a	d
c	e	f
d	e	f
e	e	f
f	f	f

Let us apply the above algorithm to the above DFA:

- P0 = {(c,d,e), (a,b,f)}
- P1 = {(c,d,e), (a,b),(f)}
- P2 = {(c,d,e), (a,b),(f)} Hence, P1 = P2.

There are three states in the reduced DFA. The reduced DFA is as follows:

Q	δ(q,0)	δ(q,1)
(a, b)	(a, b)	(c,d,e)
(c,d,e)	(c,d,e)	(f)
(f)	(f)	(f)



State Table and State Diagram of Reduced DFA

Closure Properties of Regular Languages

Union, Intersection, Difference, Concatenation, Kleene Closure, Reversal, Homomorphism, Inverse Homomorphism

Closure Properties Recall a closure property is a statementυ that a certain operation on languages, when applied to languages in a class (e.g., the regular languages), produces a result that is also in that class.

For regular languages, we can use anyυ of its representations to prove a closure property.

Closure Under Union If L and M are regular languages, so isυ L M.∪

Proof: Let L and M be the languages ofυ regular expressions R and S, respectively.

Then R+S is a regular expressionυ whose language is L M.∪

Closure Under Concatenation and Kleene Closure
Same idea:

- RS is a regular expression whose languageω is LM.
- R* is a regular expression whose languageω is L*.

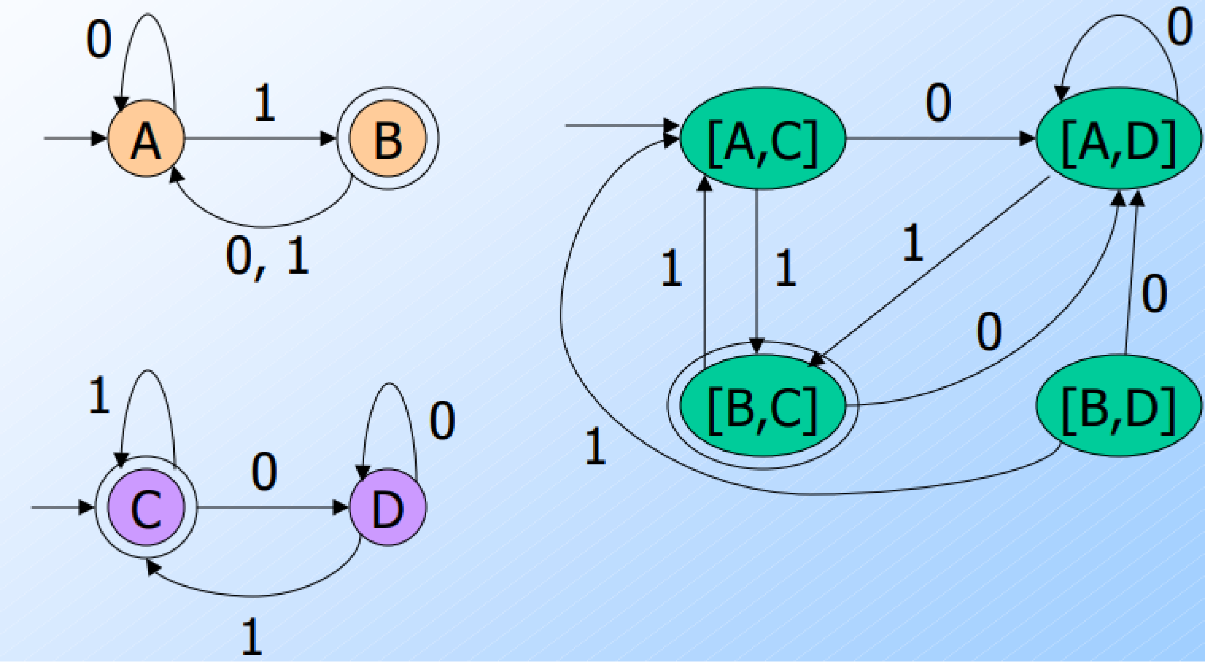
Closure Under Intersection

If L and M are regular languages, thenυ so is L M.∩

- Proof: Let A and B be DFA's whoseυ languages are L and M, respectively.
- Construct C, the product automaton of Aυ and B.

- Make the final states of C be the pairsυ consisting of final states of both A and B.

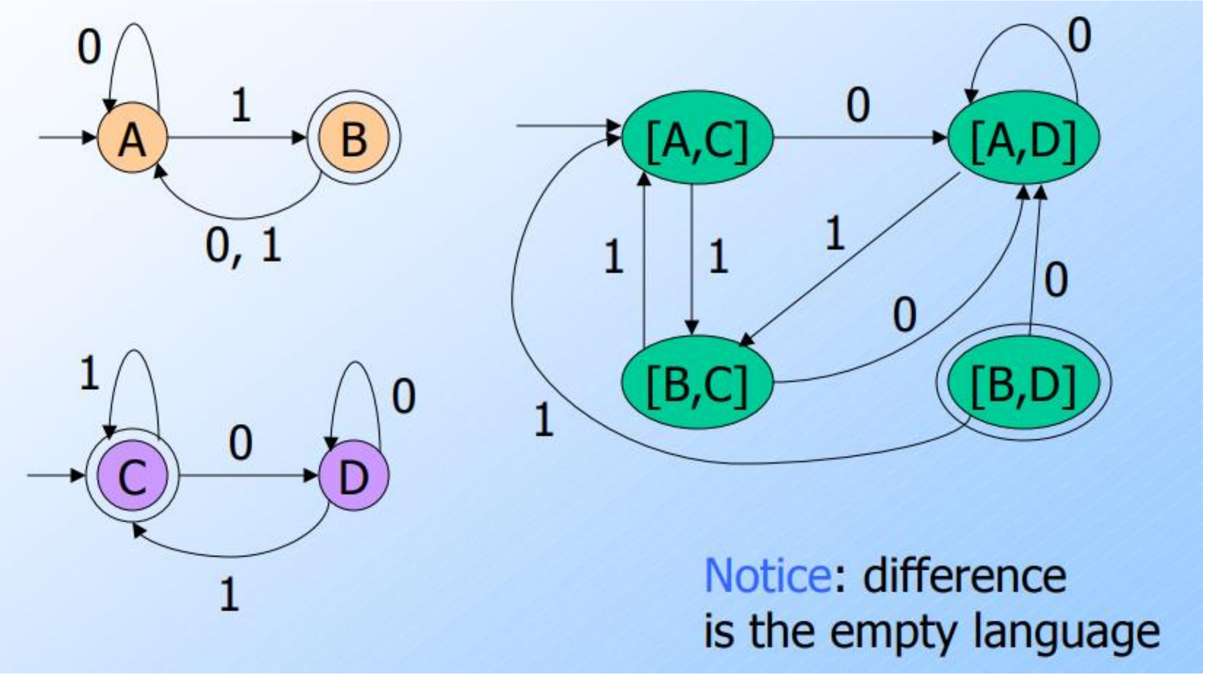
Example: Product DFA for Intersection



Closure Under Difference

- If L and M are regular languages, thenυ so is L – M = strings in L but not M.
- Proof: Let A and B be DFA's whoseυ languages are L and M, respectively.
- Construct C, the product automaton of Aυ and B.
- Make the final states of C be the pairsυ where A-state is final but B-state is not.

Example: Product DFA for Difference



Closure Under Complementation

- The complement of a language L (withυ respect to an alphabet Σ such that Σ* contains L) is Σ* – L.
- Since Σ* is surely regular, the complement of a regular language is always regular.

Closure Under Reversal

- Recall example of a DFA that acceptedυ the binary strings that, as integers were divisible by 23.
- We said that the language of binaryυ strings whose reversal was divisible by 23 was also regular, but the DFA construction was very tricky.
- Good application of reversal-closure.

Closure Under Reversal – (2)

Given language L, Lυ R is the set of strings whose reversal is in L.

Example: L = {0, 01, 100};υ L R = {0, 10, 001}.

Proof: Let E be a regular expression for L.

We show how to reverse E, to provide aυ regular expression E R for L R.

Reversal of a Regular Expression

- Basis: If E is a symbol a,υ ε, or ∅, then E R = E.
- Induction: If E is
    I. F+G, then Eω R = F R + G R.
    II. FG, then Eω R = G R F R
    III. F*, then Eω R = (F R)*.

Example: Reversal of a RE

Let E = 01* + 10*.

E^R = (01* + 10*)R = (01*)R + (10*)Rυ

= (1*)R0R + (0*)R1Rυ

= (1^R)*0 + (0R)*1υ

= 1*0 + 0*1.υ

Homomorphisms

- A homomorphism on an alphabet is aυ function that gives a string for each symbol in that alphabet.
- Example: h(0) = ab; h(1) =υ ε.
- Extend to strings by h(aυ 1…a n) = h(a 1)…h(a n).
- Example: h(01010) = ababab.υ

Closure Under Homomorphism

- If L is a regular language, and h is aυ homomorphism on its alphabet, then h(L) = {h(w) | w is in L} is also a regular language.
- Proof: Let E be a regular expression for L.υ
- Apply h to each symbol in E.υ
- Language of resulting RE is h(L)υ

Decision Properties of Regular Languages

Language classes have two important kinds of properties: 1. Decision properties. 2. Closure properties.

- Representation of Languages Representations can be formal or informal.υ
- Example (formal): represent a language byυ a RE or DFA defining it.
- Example: (informal): a logical or proseυ statement about its strings:

  1). $\{0^n 1^n \mid n$ is a non negative integer$\}$

  2). "The set of strings consisting of some number ofω 0's followed by the same number of 1's."

Decision Properties

- A decision property for a class ofυ languages is an algorithm that takes a formal description of a language (e.g., a DFA) and tells whether or not some property holds.
- Example: Is language L empty?υ

Subtle Point: Representation Matters

- You might imagine that the language isυ described informally, so if my description is "the empty language" then yes, otherwise no.
- But the representation is a DFA (or aυ RE that you will convert to a DFA). for DFA A?∅
- Can you tell if L(A) = υ

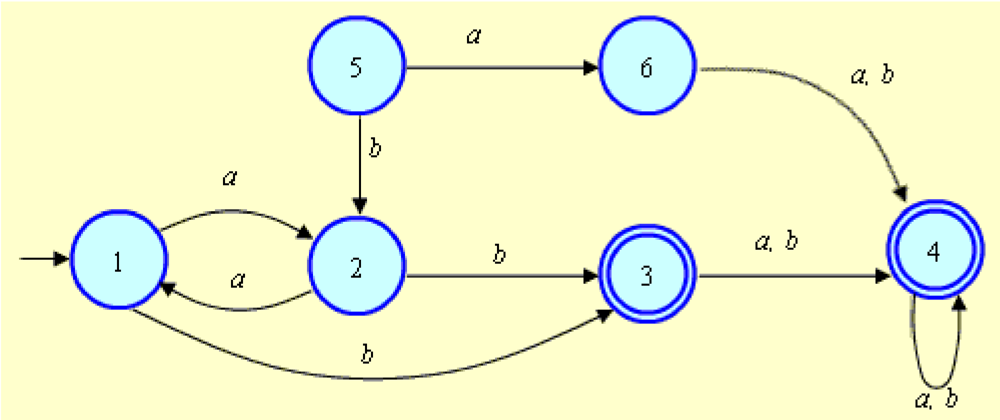
Why Decision Properties?

- When we talked about protocolsυ represented as DFA's, we noted that important properties of a good protocol were related to the language of the DFA.
- Example: "Does the protocol terminate?"υ = "Is the language finite?"
- Example: "Can the protocol fail?" = "Isυ the language nonempty?"
- We might want a "smallest"υ representation for a language, e.g., a minimum-state DFA or a shortest RE.
- If you can't decide "Are these twoυ languages the same?"
  - I. I.e., do two DFA's define the sameω language?
  - II. You can't find a "smallest."

Minimization of Automata

For any regular language L it may be possible to design different DFAs to accept L. The one with less number of states would be simpler than the other. So, given a DFA accepting a language, we further whether the DFA could further be simplified i.e. can we reduce the number of states accepting the same language.

Consider the following DFA M_1,



We can understand that it accepts the language of the regular expression

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

The same language is accepted by the following simpler DFA M_2 as well.

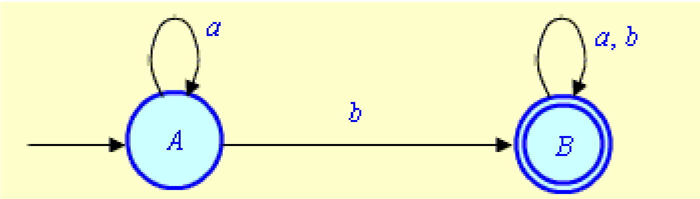


Figure 2

For any regular language L there is a unique minimal state DFA.

For any given DFA M accepting L we can construct the minimal state DFA accepting L by using an algorithm which uses following generic steps.

- First, remove all the states (of the given DFA M) which are **not** accessible from the start state i.e. sates P for which there is no string $x \in \Sigma^*$ s.t. $\hat{\delta}(q_0, x) = p$.
- Removing these states will not change the language accepted by the DFA.

- Second, remove all the trap states, i.e. all states *P* from which there is no transition out of it.
- Finally, merge all states which are "equivalent" or "indistinguishable". We need to define formally what is meant by equivalent or indistinguishable states; but at this point we assume that merging these states would not change the accepted language.

In the example, states 5 and 6 are inaccessible and hence can be removed, states 1 and 2 are equivalent and can be merged. Similarly states 3 & 4 are also equivalent and can be merged together to have the minimal DFA M_2 as produced above.

To construct the minimal DFA we need to see how to find out indistinguishable or equivalent states for merging.

we start with a definition and then proceed to find method to construct minimal state DFAs.

Properties

i) (P,Q) are two states equivalent states if

* $\delta(P,w) \in F \rightarrow \delta(Q,w) \in F$

* $\delta(P,w)$ '$\notin$' $F \rightarrow \delta(Q,w)$ '$\notin$' F

Property 2: $|w| = 0$, it is 0 equivalent

$|w| = 1$, it is 1 equivalent

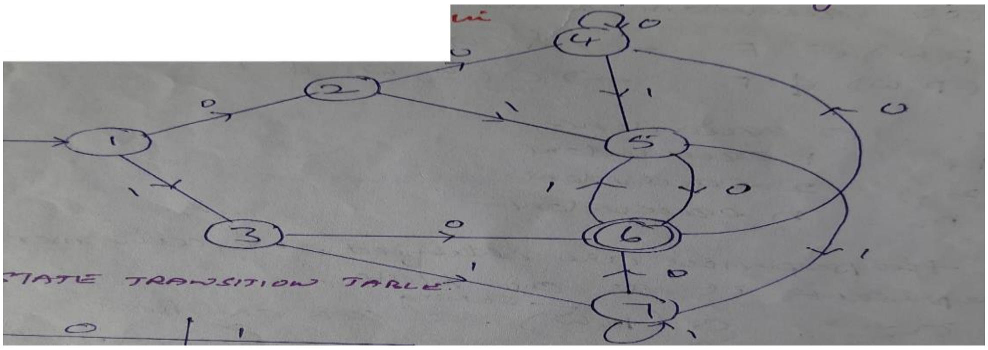
$|w| = 2$, it is 2 equivalent

$|w| = n$, it is n equivalent

If these properties are satisfied, we can merge P,Q states and represent it as a single state.

Partitioning Method

Question: Find the minimum state FA for the given machine



Step 1: Construct Transition table

δ	0	1
→1	2	3
2	4	5
3	6	7
4	4	5
5	6	7
*6	4	5
7	6	7

Step 2: Applying Partitioning algorithm

Partition P1 = { {1,2.3,4,5,7}, {6}

Construct P2 = by portioning P1 subsets

P2 = {{ 1, 2, 4}, { 3, 5, 7}, {6}

Construct P3 by partitioning P2 subsets

P3 = {{ 1, 2, 4}, { 3, 5, 7}, {6}

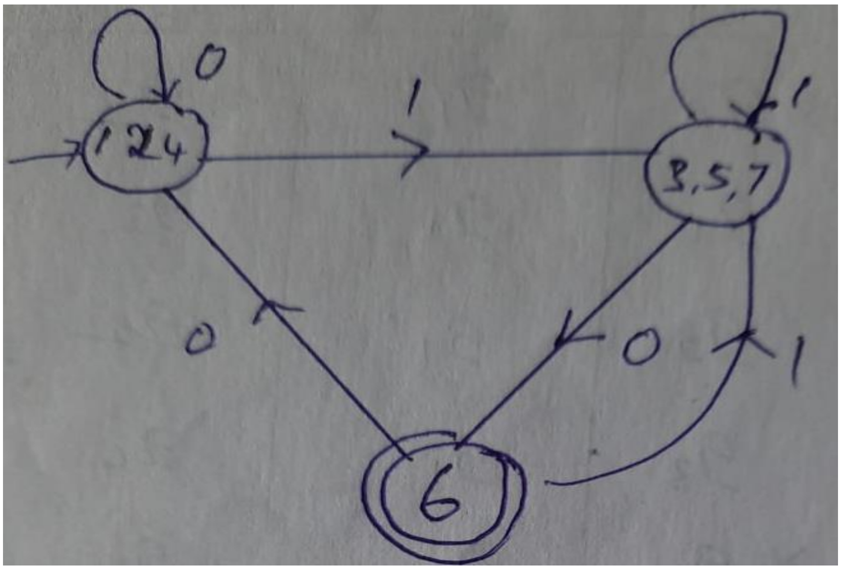
The process is stopped when $p_i = p_i - 1$.

Here, P2 = P3 so, Process is stopped.

Step 3: New Transition Table

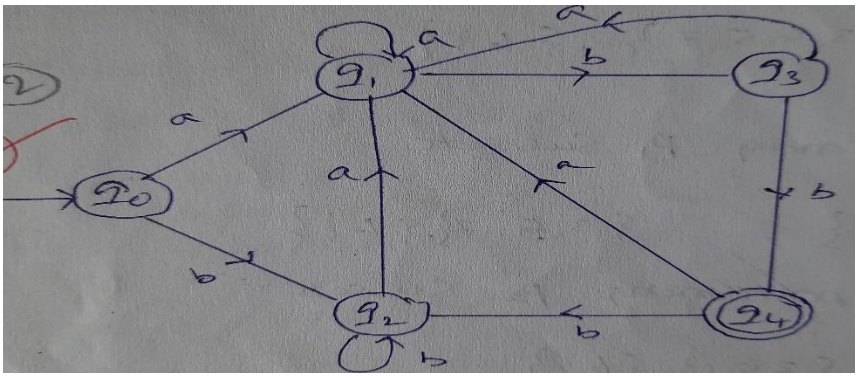
δ	0	1
[124]	[124]	[357]
3	[6]	[357]
6	[124]	[357]

Step 4: Minimal DFA

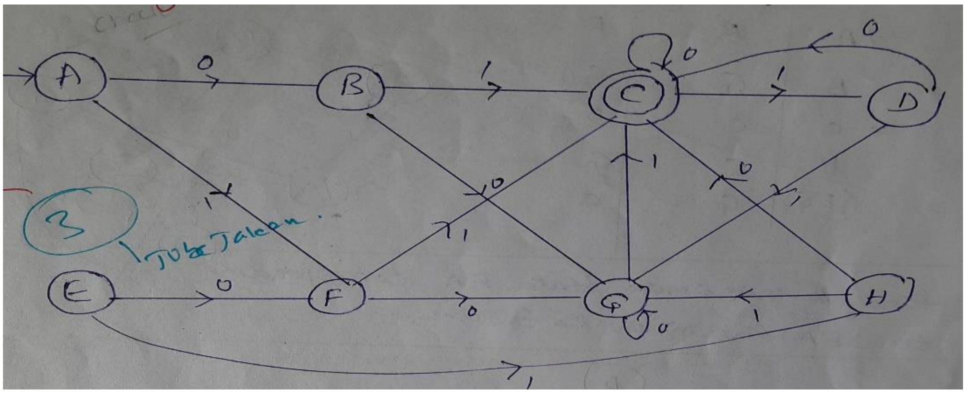


Exercise

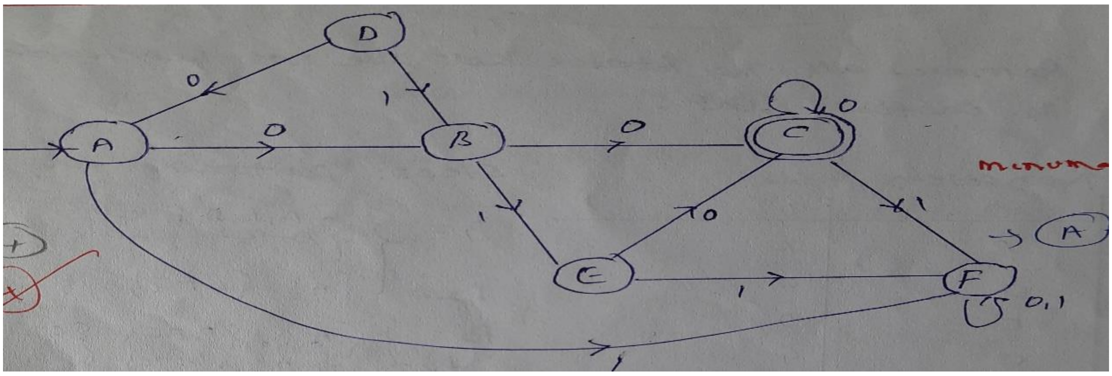
2. Find the minimal DFA



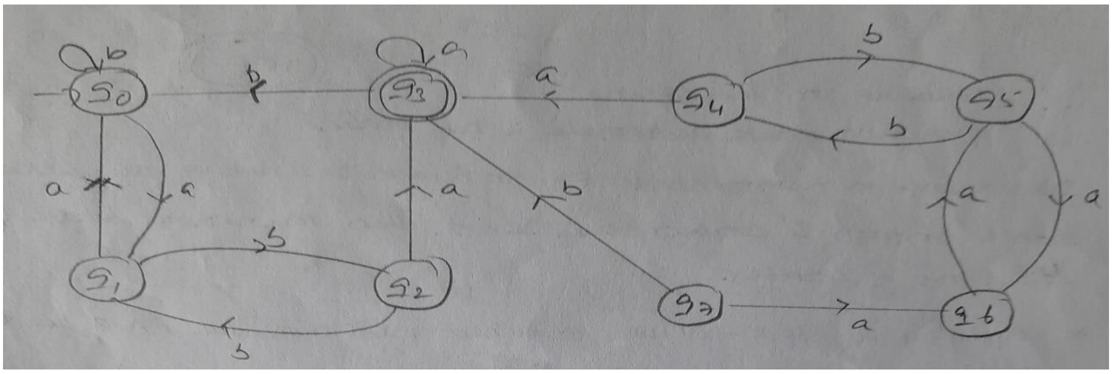
3. Find the minimal DFA



4. Find the minimal DFA



5. Find the minimal DFA



=====================