Formal Languages & Autometa Theory

Lecture Material on
REGULAR EXPRESSIONS

PNRL Chandra Sekhar

Department of Computer Science & Engineering GST, GITAM

August 9, 2023

The following material will give you an overview of what you learn in the class

1 Introduction

A regular expression (shortened as regex or regexp), sometimes a rational expression, is a sequence of characters that specifies a match pattern in text. Usually, such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings or for input validation.

The concept of regular expressions began in the 1950s when the American mathematician Stephen Cole Kleene formalized the concept of a regular language. They came into everyday use with Unix text-processing utilities. Different syntaxes for writing regular expressions have existed since the 1980s, one being the POSIX standard and another widely used being the Perl syntax.

Regular expressions are used in search engines, search and replace dialogue of word processors and text editors, text processing utilities such as SED and AWK, and in designing lexical analysis. Regular expressions are supported in many programming languages.

Regular expression techniques are developed in theoretical computer science and formal language theory. They have the same expressive power as regular grammar.

The language accepted by finite automata can be easily described by simple expressions called Regular Expressions. It is the most effective way to represent any language. The languages accepted by some regular expressions are referred to as Regular languages. A regular expression can also be described as a sequence of patterns that defines a string.

Definition: Let Σ be an alphabet then the Regular Expression over Σ can be recursively defined as follows

- ϵ is a Regular Expression indicating the language containing an empty string. (L(ϵ) = ϵ)
- ϕ is a Regular Expression denoting an empty language. (L(ϕ) = { })
- for each a in Σ, a is a Regular Expression where L = {a}
- If r is a Regular Expression denoting the language L(r) and s is a Regular Expression denoting the language L(s), then

  r + s is a Regular Expression corresponding to the language L(r) ∪ L(s) where L(r+s) = L(r) ∪ L(s).

  r . s is a Regular Expression corresponding to the language L(r) . L(s) where L(r.s) = L(r) . L(s)

  R* is a Regular Expression corresponding to the language L(R*)where L(R*) = (L(R))*
- , if r is a regular expression, then (r) is also a regular expression.

  The precedence order from higher to lower as follows (), *, .,+

If we apply any of the above five rules several times, they are Regular Expressions.

Examples of Regular Expressions – corresponding Regular Set

1. (0 + 10*)– L = { 0, 1, 10, 100, 1000, 10000, . . . }
2. (0*10*) – L = {1, 01, 10, 010, 0010, . . .}
3. (0 +)(1 +)– L = {ϵ, 0, 1, 01}
4. (a+b)* – L = {ϵ, a, b, aa , ab , bb , ba, aaa. . . . . . .} (Set of strings of a's and b's of any length including the null string)
5. (a+b)*abb – L = {abb, aabb, babb, aaabb, ababb, ...} (Set of strings of a's and b's ending with the string abb.)
6. (11)* – L= {ϵ, 11, 1111, 111111, ...} (Set consisting of even number of 1's including empty string)
7. (aa)*(bb)*b –L = {b, aab, aabbb, aabbbbb, aaaab, aaaabbb, ...} (Set of strings consisting of even number of a's followed by odd number of b's)
8. (aa + ab + ba + bb)* –L = {aa, ab, ba, bb, aaab, aaba, ...} (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)
9. (1+10)* – L={1,10,11,111,....} (set of all strings that begins with 1 and ends with either 1 or 0)
10. 0*1*2* – L={$0^m 1^n 2^k | m, n, k >= 0$} (set of strings containing any no of 0's followed by any no of 1's followed by any no of 2's)

1.1 Identities Related to Regular Expressions

Two regular expression R and S are equivalent iff their corresponding languages are same i.e. L(R) = L(S).

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

a) $\phi^* = \epsilon$

b) $\epsilon^* = \epsilon$

c) $R + \phi = \phi + R = R$ (The identity for union)

d) $R\epsilon = \epsilon R = R$ (The identity for concatenation)

e) $\phi R = R\phi = \phi$ (The annihilator for concatenation)

f) R + R = R (Idempotent law)

g) RR* = R*R

h) R*R* = R*

i) (R*)* = R*

j) (PQ)*P =P(QP)*

k) (a+b)* = (a*b*)* = (a*+b*)* = (a+b*)* = a*(ba*)*

l) L(M + N) = LM + LN (Left distributive law)

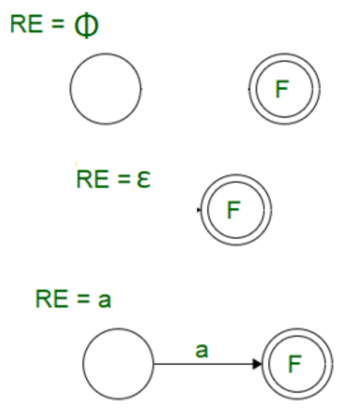
m) (M + N)L = ML + NL (Right distributive law)

n) $\epsilon + RR* = \epsilon + R * R = R*$

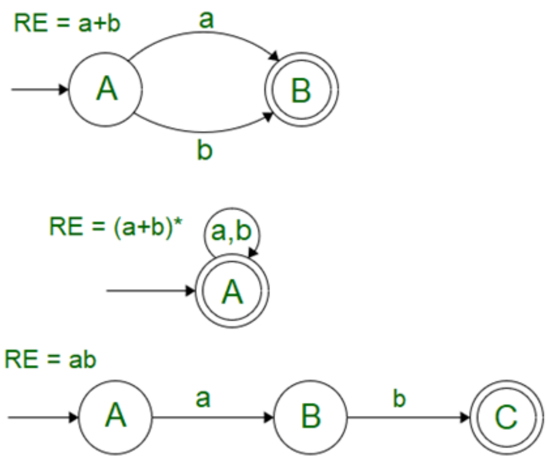
2 Equivalence of FA & RE

A language is said to be regular if it can be represented using Finite Automata or if a Regular Expression can be generated from a given Finite Automata to recognize the language. This definition can lead to "For every Regular Expression corresponding to the language, a Finite Automata can be generated."

For the primitive operations over regular expressions like:- $\phi, \epsilon, a, a+b, ab, (a+b)*$, It's pretty easier to make the Finite Automata by intuition, as shown below.



For certain expressions like :- (a+b), ab, (a+b)* ; It's fairly easier to make the Finite Automata by just intuition as shown below.



The problem arises when we are provided with a longer Regular Expression. This brings the need for a systematic approach towards Finite Automata generation, which Kleene has put forward in Kleene's Theorem – I.

2.1 Equivalence of Regular Expression to Finite Automata

Kleene's Theorem-I For any Regular Expression r representing Language L(r), a Non-Deterministic Finite Automata with ϵ - moves that accept the same language exists. From the basic definition of regular definition as it is true for primitive operations, from them we can define $\epsilon - NFA$ in three different cases. These can be used to define other forms of regular expressions reclusively.

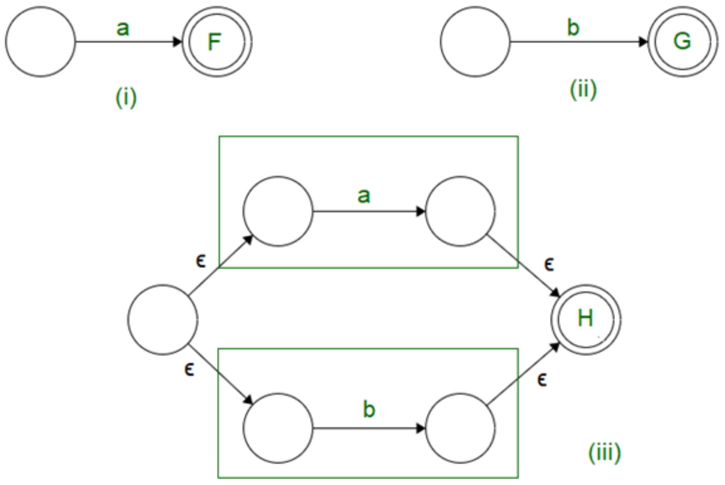
If r is a Regular Expression denoting the language L(r) and s is a Regular Expression denoting the language L(s), then

case 1 - r + s is a Regular Expression corresponding to the language L(r) ∪ L(s) where L(r+s) = L(r) ∪ L(s).

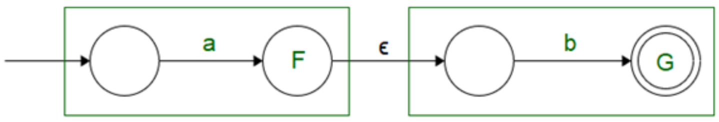
case 2 - r . s is a Regular Expression corresponding to the language L(r) . L(s) where L(r.s) = L(r) . L(s)

case 3 - R* is a Regular Expression corresponding to the language L(R*)where L(R*) = (L(R))*

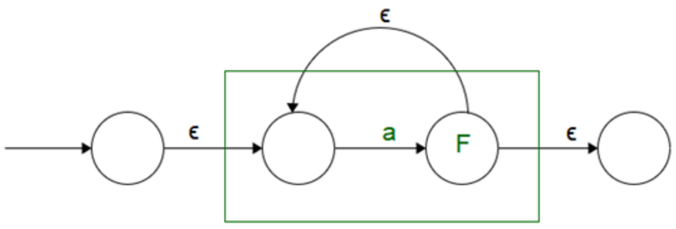
Let S accept L = a and T accept L = b, then R can be represented as a combination of S and T using the three cases as follows: Case 1: R = S + T



Case 1: R = S . T



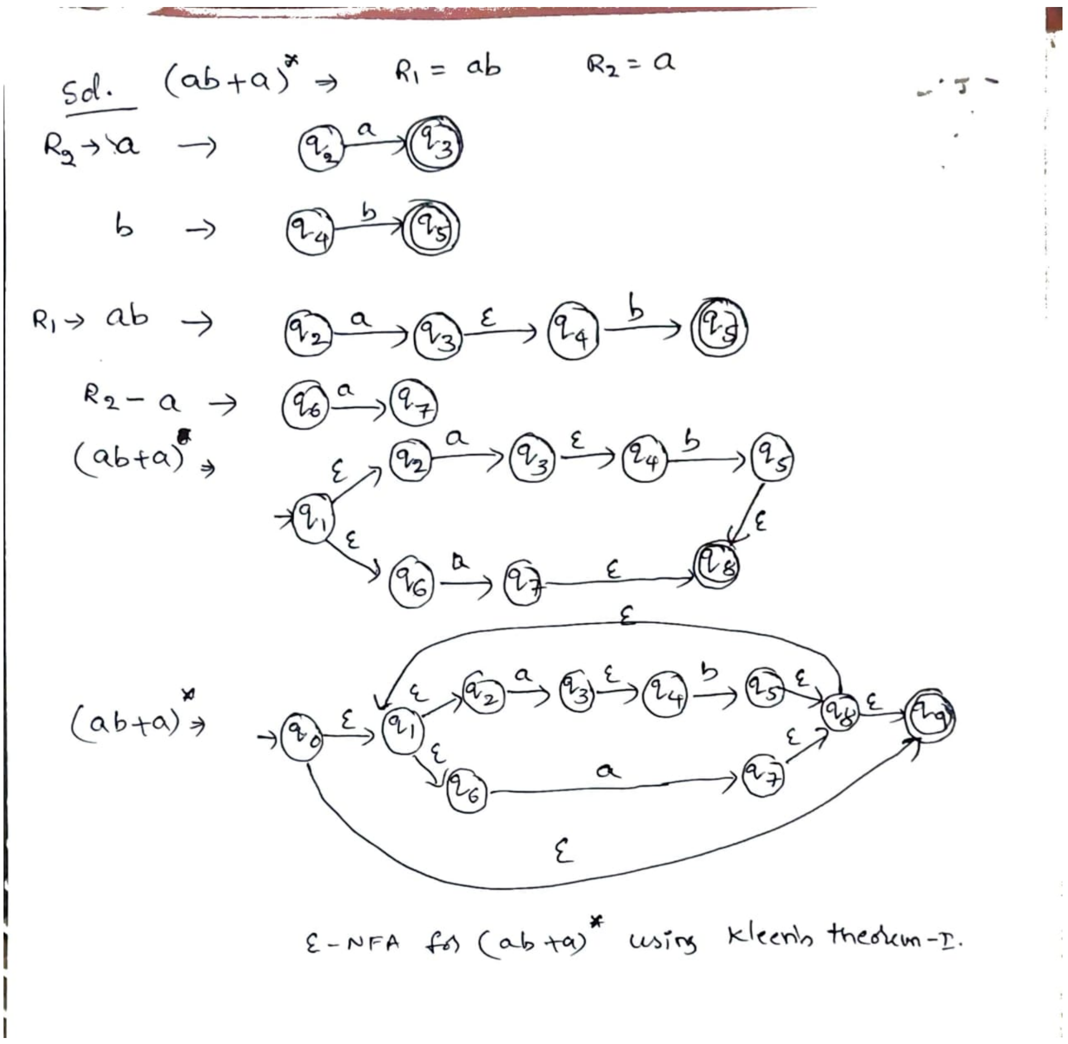
Case 1: R = S*



Problem 1

Make a Finite Automata for the expression (ab+a)*

Solution.


Sol. (ab+a)* → R1 = ab R2 = a
R2 → a →
b →
R1 → ab →
R2 − a →
(ab+a)* →
(ab+a)* →
ε-NFA for (ab+a)* using Kleen's theorem-I.

2.2 Equivalence of Finite Automata to Regular Expression

The 2nd part of Kleen theorem states that the language accepted by a Finite Automata is regular. i.e., a string is in Finite Automata M is accepted only if there is a path from the initial state to the final state. In the Finite Automata

- the serial path corresponds to concatenation
- the parallel path corresponds to the union
- the loop corresponds to Kleen closure operations

then we can find out the complete regular expression for the path from the initial state to final state. Thus all strings in the regular language are generated through regular expression, the correspondence between Finite Automata and Regular expression is established. i.e., a mechanism to generate regular expressions corresponding to given Finite Automata is established and it is known as ardens theorem.

Theorem. ***Arden's Theorem**: If P and Q are two regular expressions over Σ, and if P does not contain ϵ, then the equation in R given by R = Q + RP has a unique solution i.e., R = QP*.*

That means, whenever we get any equation in the form of R = Q + RP, then, we can directly replace it with R = QP*. In our proof, first prove that R = QP* is a solution and then prove it is the unique solution.

Proof. Let P and Q are two regular expressions over Σ

if p not contained ϵ the $\exists$ R such that

R = Q + RP ......(i)

Now, replacing R by R = QP* in (i), we get,

$$\begin{aligned} RHS : R &= Q + QP*P \\ R &= Q(\epsilon + P*P)R \\ &= QP* \end{aligned}$$

(Taking Q as common and since ϵ + R*R = R*)

Hence proved. Thus, R = QP* is the solution of the equation R = Q + RP.

Now, we have to prove that this is the only solution to this equation. Let me take the LHS of equation (i) and replace R by R = Q + RP,

$$\begin{aligned} R &= Q + (Q + RP)P \\ &= Q + QP + RP^2 \end{aligned}$$

Again, replace R by R = Q + RP:-

$$\begin{aligned} R &= Q + QP + (Q + RP)P^2 \\ &= Q + QP + QP^2 + RP^3 \\ &= ... \\ &= ... \\ &= Q + QP + QP^2 + .. + QP^n + RP^{(n+1)} \end{aligned}$$

Now, replace R by R = QP*, we get,

$$R = Q + QP + QP^2 + .. + QP^n + RP^{(n+1)}$$

Taking Q as common,

$$\begin{aligned} R &= Q(\epsilon + P + P^2 + .. + P^n) + RP^{n+1} - - - -(ii) \\ &= QP^* + RP^{n+1} - - - - - (iii) \end{aligned}$$

$[\epsilon + P + P^2 + .. + P^n$ represent the closure of P]

For eq. (iii) the possible solutions are $R = QP*$ or $R = RP^{n+1}$

since $P \neq \epsilon$ and $i \geq 0 \implies |R| < |RP^{n+1}|$ i.e., $|RP^{n+1}|$ has no string whose length $<$ n+1 and it can't be a solution for R.

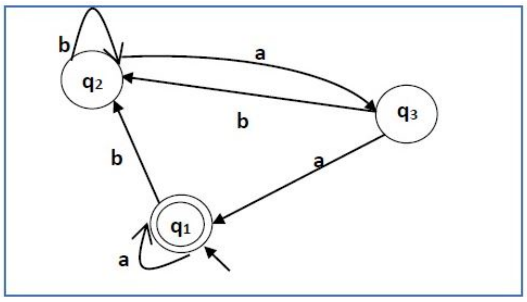
Hence proved that the only feasible solution is R = QP* and thus QP* is the unique solution of the equation R = Q + RP.

Note : Arden's theorem is used to convert given finite automata to a regular expression.

□

Problem 2

Construct a regular expression corresponding to the following finite automata:



Solution. Here the initial state and final state is q1.

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

q1 = ϵ + q1a + q3a (ϵ - move is because q1 is the initial state)

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

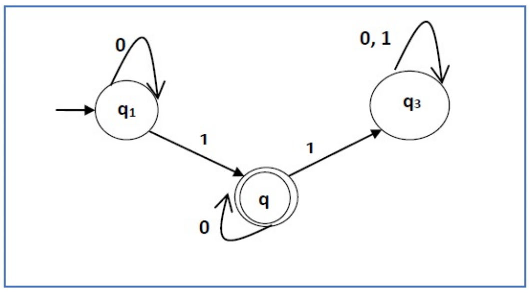
= q1a + q2aa + ϵ (Substituting value of q3)

= 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 3

Construct a regular expression corresponding to the following 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*.

3 Closure Properties

The closure property relates to a mathematical system that states that if a set of elements is closed under some operation, performing that operation on any two elements in the set results in the same set.

For ex, the sum of two natural numbers results in a natural number, and the subtraction of two integers is always an integer. In contrast, an integer divided by another does not give an integer. i.e., natural nos are closed under addition and subtraction, whereas integers are not closed under division. Similarly, Matrices are closed under addition and subtraction but not on determinant.

The closure property is directly linked with such properties of any given set concerning an operation.

Definition The closure property in regular languages is defined as certain operations on regular languages which are guaranteed to produce regular language.

As we know, every regular set can be expressed by its corresponding regular expression, and for every regular expression, there exists a corresponding regular set.

If an operation performed on regular sets leads to another set that is also regular, then the regular sets are closed over that operation.

Regular languages are closed under the following operations:

- union
- concatenation
- Kleen closure
- intersection
- complement
- set-difference
- reverse
- homomorphism

Theorem. *Regular Sets are closed under basic operations: union, concatenation and kleen closure.*

Proof. If L1 and L2 are two regular languages the ∃ regular expression R1, R2 such that, L1= L(R1) and L2 = L(R2).

1. $L1 \cup L2 = L(R1) \cup L(R2) = L(R1 + R2) \implies L1 \cup L2$ regular
2. $L1.L2 = L(R1).L(R2) = L(R1.R2) \implies L1.L2$ regular
3. If R1 is a RE then R1* is also RE. So, L1* = L(R1*) = L(R1)* $\implies$ L1* is regular. □

Theorem. *Regular Sets are closed under complementation and intersection operations.*

Proof. If L is regular, there is a DFA $M = (Q, \Sigma, , q0, F)$ such that L = L(M) $\subseteq \Sigma^*$.

To accept $\overline{L}$ i.e., $\Sigma^* - L$, the machine must complement the final states of M

i.e., $M^{'} = (Q, \Sigma, , q_0, \{Q - F\})$ where all non-final states becomes the final states and $M^{'}$ accepts set of all strings which are in $\Sigma^* - L \implies \overline{L}$ accepted by $M^{'}$ is a regular language.

∴ regular languages are closed under complementation.

From algebra, $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

Since, L1 & L2 are two regular sets, their complementations say $\overline{L_1}$ and $\overline{L_2}$ are regular then their union $\overline{L_1} \cup \overline{L_2}$ also regular and its complementation $\overline{\overline{L_1} \cup \overline{L_2}}$ is also regular.

∴L$_1 \cap L_2$ is also regular □

Theorem. *Regular Sets are closed under trasposition (reverse) opation.*

Proof. If L is regular, there $\exists$ a DFA $M1 = (Q, \Sigma, , q0, F)$ such that L_1 = L(R1) = L(M1).

For every string $s \in L$ there a path in M from its initial state to final state.

Not to accept the reverse of w = w^R we create a DFA M2 as follows:

1. Reverse the directions of arcs.
2. Change the initial state as final state.
3. Create a new initial state q and connect it to all the final states of M1 using ϵ-transitions.
4. Obtain the equivalent DFA for M2.

Let L_2 be the set of all states accepted by M2 where every string $x \in L_2$ is the reverse of some string $w \in L_1$. Thus L_2 is a set of transposed strings of L_1.

Since L_2 is associated with a DFA M2, it is associated with a regular language.

$\therefore$ regular languages are closed under transposition. □

4 Pumping lemma for Regular languages

It is clear that if any language is accepted by some finite automata or expressed by regular expression then the language is said to be regular. Every language is not regular. so, it is important to know which languages are regular and which are not? The pumping lemma is a powerful mechanism for proving certain language are not regular. It is a negative test theorem in the sense that it can't be used to prove as regular but certainly it can be used to prove certain languages are not regular.

Theorem. *For any regular language L, $\exists$ an integer n, such that for all $x \in L$ with $|x| \geq n, \exists u, v, w \in \Sigma^*$, such that x can be decomposed into uvw where (1) $|uv| \leq n(2)|v| \geq 1$. Then for all, $i \geq 0 : uv^iw \in L$.*

In other words, if a string v is 'pumped,' i.e., if v is inserted any number of times, the resultant string remains in L.

4.1 Application of pumping lemma

Pumping lemma is used to show that certain languages are not regular.

It should never be used to show a language is regular.

If L is regular, it satisfies the Pumping lemma.

If L does not satisfy the Pumping Lemma, it is not regular.

Steps to prove that a language is not regular by using pumping lemma are as follows

- Assume that L is regular, then PL holds for L
- Let the pumping length be P where all strings longer than P can be pumped —w—≥P
- Find a string w ∈ L such that —w—
- Decompose/divide w into xyz such that $|xy| \leq n\&|y| \geq 1$
- for some i, show that $xy^iz \notin L$.
- It means that none of these can satisfy all the pumping conditions at the same time.

- Hence conclude that w can not be pumped, and we arrive at a contradiction to our assumption. So, L is not regular.

Problem 4

Prove that L $=\{0^n1^n | n \geq 0\}$ is not regular

Solution. Let us assume that L is regular, then by Pumping Lemma, the above-given rules follow.
Now, let w = $0^n1^n \in$ L and —w— = n+n =2n $\geq$ n.
By Pumping Lemma, decompose w into xyz such that $|xy| \leq n \& |y| \geq 1$.
Then we need to find i such that $xy^iz \notin L$ to get a contradiction.
i.e., w = 0^n1^n and decompose into xyz as x = $0^{n-1}, y = 0^1, z = 1^n$ where —xy—=—$0^{n-1}0^1$—=n-1+1=n $\leq$ n and —y— ≥ 1
When pump y 0 times then,
$xy^0z = 0^{n-1}0^01^n = 0^{n-1}1^n \notin L$
Which is a contradiction to our assumption.
Hence it is proved that L is not regular.

Problem 5

Show that L = $\{a^p$ — p is a prime no$\}$ is not regular.

Solution. Assume L is regular and let n be no of states in FA accepting strings of L.
Let p be a prime no greater than n and let w = $a^p \implies |w| = p$ —— (1)
We can decompose w into xyz by pumping lemma with $|xy| \leq n \& |y| \geq 1$ where xyz are strings of a's.
Let $y = a^m$ where —y—=m for some $m \geq 1 (\& \leq n)$ ——(2)
Let i = p+1, —–(3)
then

$$\begin{aligned}|xy^iz| &= |xyz| + |y^{i-1}| \\ &= p + (i-1).m ------ by(1\&2) \\ &= p + p.m ------- by(3) \\ &= p(1+m)\end{aligned}$$

By pumping lemma, $xy^iz \in L$ but, $|xy^iz| = p(1+m)$ is not a prime no.
So, $xy^iz \notin L$. Hence it is proved that L is not regular.

5 Minimization of DFA

DFA minimization stands for converting a given DFA to its equivalent DFA with minimum number of states. i.e., reduce the no of states from the given DFA by removing all redundent state. Redundant states are nothing bus equivalent states and they can be cobined and make them as single state.

When can we say two states are equivalent?
When the transitions on both states are going either to final states or non-final states on the same input symbol.
i.e., if σ(p,a) takes to some final state and σ(q,a) also takes to a final state (not necessarily same) or if σ(p,a) takes to some non-final state and σ(q,a) also takes to non-final state, then, p & q are said to be equivalent or distinguishable.

DFA minimization is also called as Optimization of DFA and uses partitioning algorithm.

Steps to obtain Minimization of DFA:
Let there is a DFA $M(Q, \Sigma, q0, \delta, F)$ which recognizes a language L. Then the minimized DFA $M'(Q', \Sigma, q0, \delta', F')$can be constructed for language L as:

a) Create a partition set π_0 from Q (Zero equivalent) as set of non-final states and final states. $\pi_0 = \{Q_1^o, Q_2^o\}$ where Q_1^0, Q_2^0 are set of non-final and final states.

b) initialize k=1

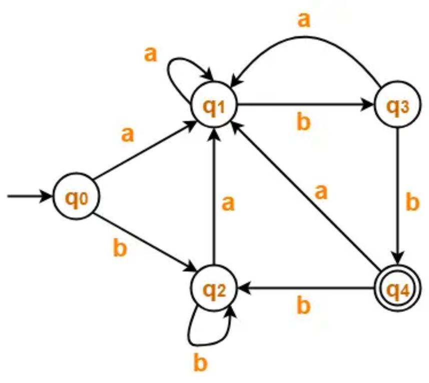
c) Construct the partition π_{k+1} from π_k.
Let Q_i^k be any subset in π_k and let q_1, q_2 are two states in Q_i^k and if $\delta(q_1, a)\&\delta(q_2, a)$ gives the states in same subgroup of π_k then $q_1\&q_2$ are equivalent/distinguishable and keep them in the same group otherwise separate them.
repeat the above step for every pair of states in Q_i^k of π_k and obtain all the groups of states in π_{k+1}

d) repeat stops until there is no change in the partitions (i.e., $\pi_k = \pi_{k+1}$)

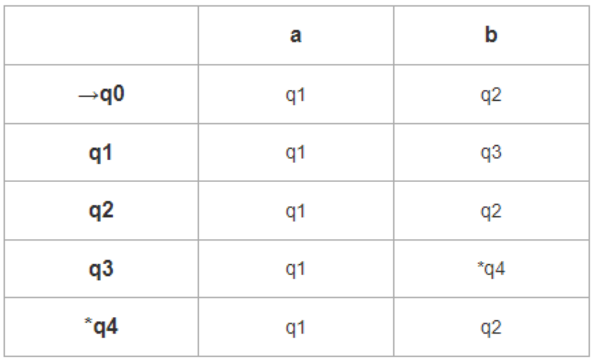
e) Replace all the equivalent states in each partition merge them into one state for the minimized DFA.

Problem 6

Minimize the following DFA.

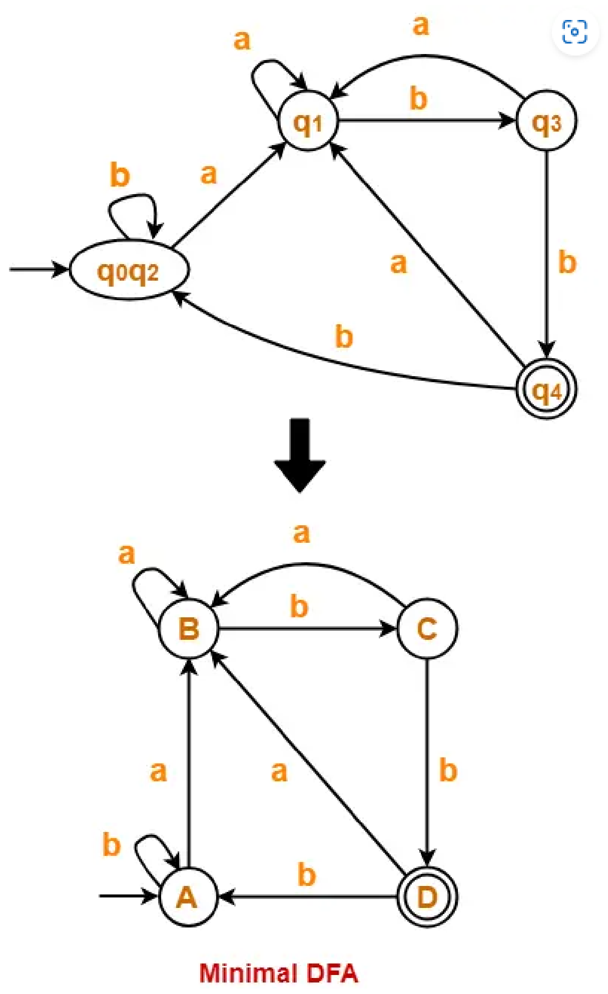


Solution. 1. The given DFA contains no dead states and inaccessible states, so no need to remove any states.
2. Draw a state transition table



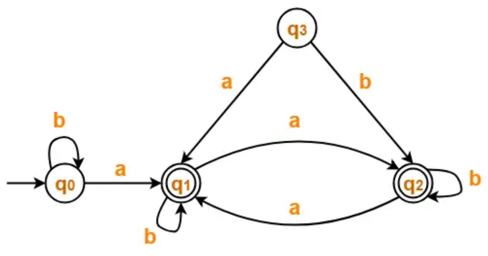
	a	b
→q0	q1	q2
q1	q1	q3
q2	q1	q2
q3	q1	*q4
*q4	q1	q2

3.Now apply Equivalence Theorem / partitioning algorithm, to the given DFA
π_0 = q0 , q1 , q2 , q3   q4 - partition the non-final and final states
π_1 = q0 , q1 , q2   q3   q4 - $q_2\&q_3$ are not equivalent, separate them
π_2 = q0 , q2   q1   q3   q4 - $q_1\&q_2$ are not equivalent, but $q_1\&q_2$ are equivalent. So, separate q_1 as new partition
π_3 = q0 , q2   q1   q3   q4 - no change in the partition
Since $\pi_3 = \pi_2$, so the iteration of the partitioning stops.
From π_3, we infer that states q0 and q2 are equivalent and can be merged together.
So, the obtained minimal DFA is



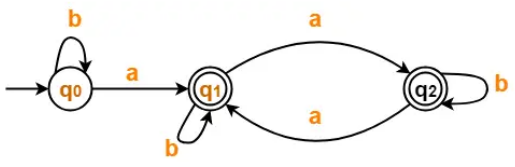
Problem 7

Minimize the following DFA.

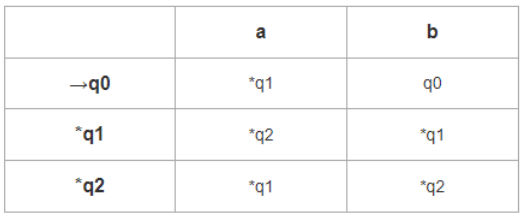


Solution. 1. q3 is inaccessible from the initial state. So, eliminate it and its associated transitions from the DFA.

The resulting DFA is-

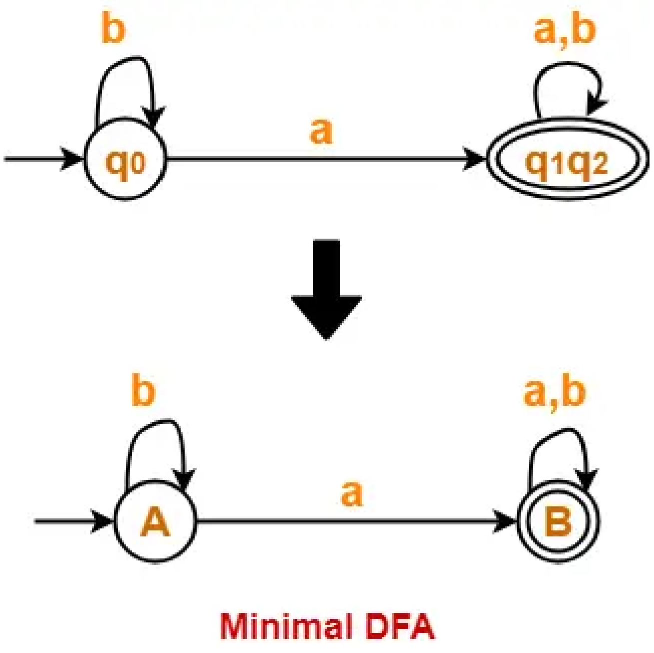


2. The state transition table is:



	a	b
→q0	*q1	q0
*q1	*q2	*q1
*q2	*q1	*q2

3. Now apply Equivalence Theorem / partitioning algorithm, to the given DFA
$\pi_0 =$ q0 q1 , q2
$\pi_1 =$ q0 q1 , q2
Since $\pi_1 = \pi_0$, so the iteration stops.
From π_1, the states q1 and q2 are equivalent and can be merged together.
So, the resultant minimal DFA is:


b
a,b
q0
a
q1q2
b
a,b
A
a
B
Minimal DFA