Regular language :

A non regular language **:** **1)requires memory**
2) is not recognized by any final state machine (FSM)

Operations on regular languages :

Ex: A = {pq, r } and B = {t, uv}

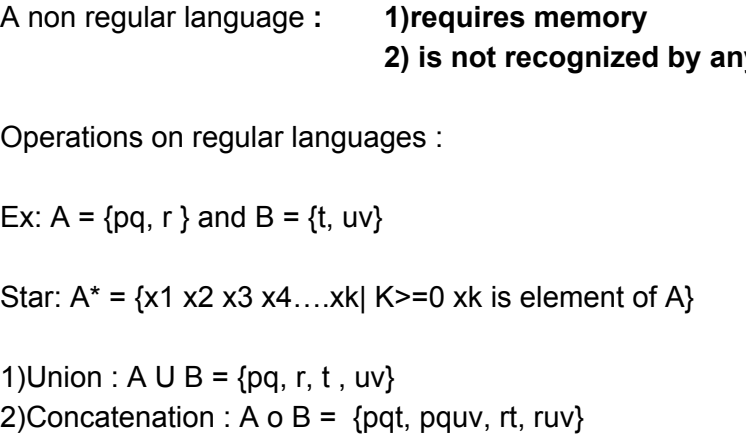
Star: A* = {x1 x2 x3 x4….xk| K>=0 xk is element of A}

1)Union : A U B = {pq, r, t , uv}
2)Concatenation : A o B = {pqt, pquv, rt, ruv}
3)Star: A* = { ε, pq, r, pqr, rpq, pqpq, rr, pqpqpq,rrr …. }
Theorem 1: The union of 2 regular languages is also a regular language
Theorem 2: The concatenation of 2 regular languages is also a regular language

PREREQUISITES:

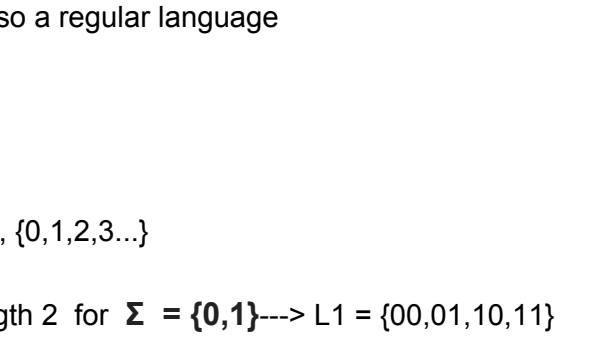
-Symbols
-Alphabet : denoted by **Σ,** collection of symbols. Ex : {a,b} , {0,1,2,3...}
-String: Sequence of symbols . Ex: a,b,b,a
-Language: Set of strings / Ex: L1 = Set of all strings of length 2 for **Σ = {0,1}**---> L1 = {00,01,10,11}
-Powers of **Σ :** Ex: **Σ^0 = set of strings of length 0 = {ε} , Σ^2 = set of strings of length 2 = {00,01,10,11}...**
-Cardinality: number of elements in a set. Ex: Cardinality of **Σ^2 is 4. -----> Cardinality = 2^n for alphabet of 2 symbols.**
-Σ^* = set of all possible strings of all length over {0,1}----> it's infinite

DFA:



	0	1
A	C	B
B	D	A
C	A	D
D	B	C

$\delta^*(q_0, ab) = q_2$



Observation: There is a walk from q to q' with label w

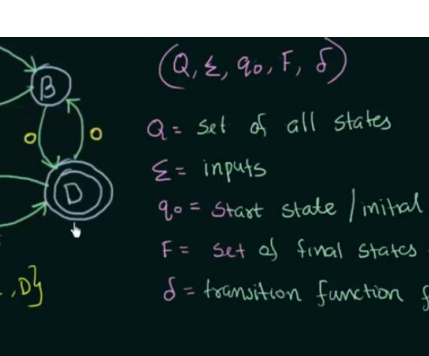
$\delta^*(q, w) = q'$

$w = \sigma_1\sigma_2\cdots\sigma_k$

Another Example

The language is regular: $L = \{awa : w \in \{a,b\}^*\}$

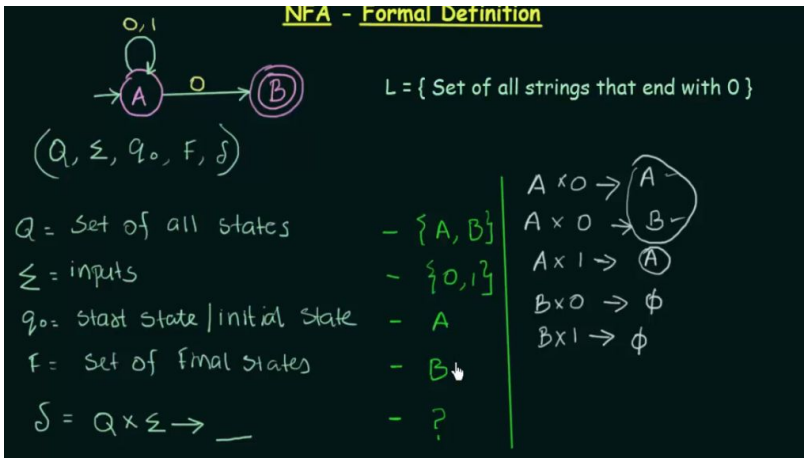
$L = L(M)$

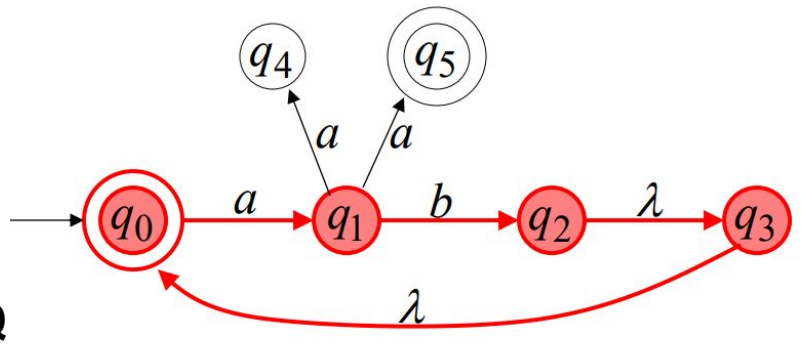


w is a walk, which means it could be anything

NFA:

$$\delta^*(q_0, ab) = \{q_2, q_3, q_0\}$$





Dell = 2^Q

Formally

$q_j \in \delta^*(q_i, w)$: there is a walk from q_i to q_j with label w

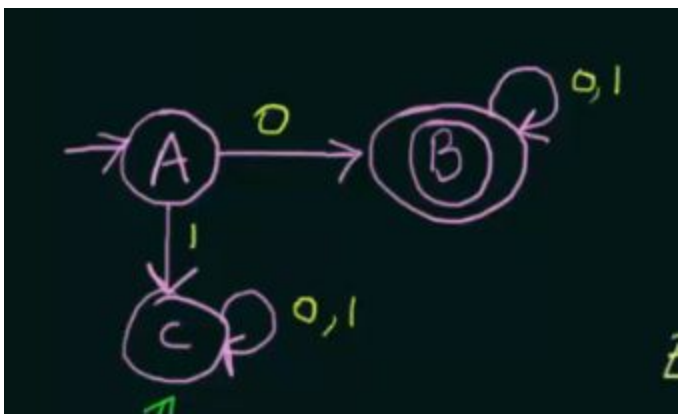
$F = \{q_0, q_5\}$

$\delta^*(q_0, ab) = \{q_2, q_3, \underline{q_0}\}$ $\in F$ $\quad ab \in L(M)$

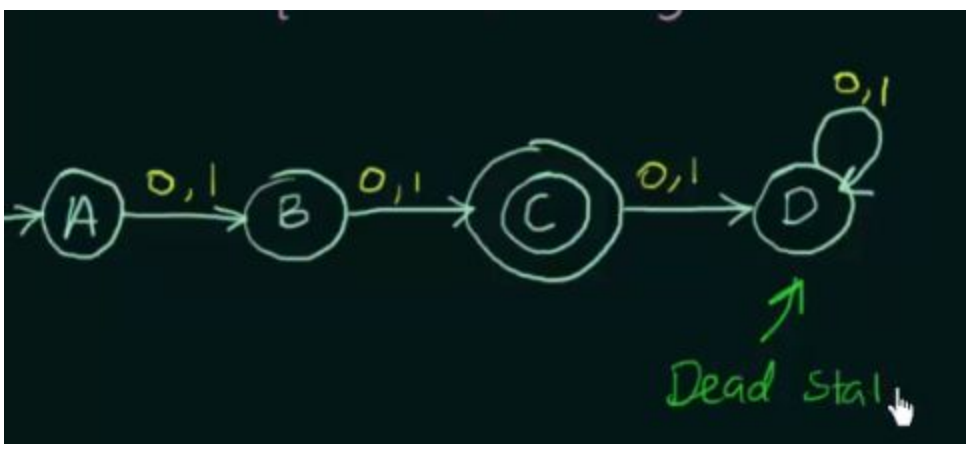
Deterministic Finite automata:

-1 unique next state for particular input
- no choices/ randomness
-given the current state we know the next state

EXAMPLE 1: L1 = SET OF ALL STRINGS THAT START WITH 0



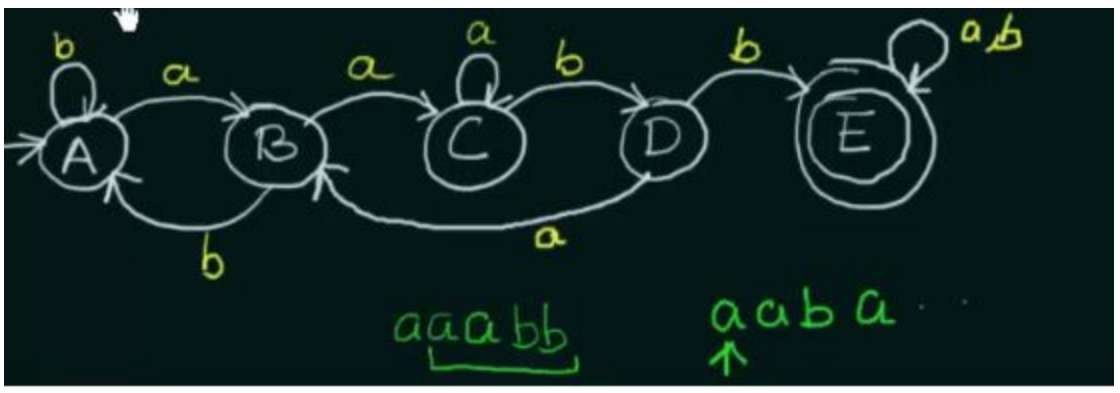
EXAMPLE 2: L2 = SET OF ALL STRINGS OF LENGTH 2



EXAMPLE 3: L3 = SET OF STRINGS OVER **Σ=**{a,b} THAT DOES NOT CONTAIN THE STRING aabb IN IT.

=> WRITE L3* = SET OF ALL STRINGS THAT CONTAIN aabb !!!

L3* =

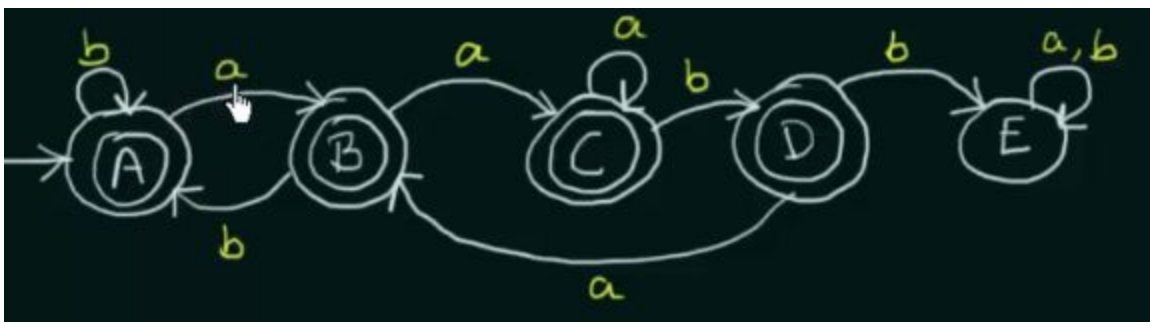


from state D, if input is 'a' then it goes back to the position of the first 'a', which is @ state B (upon first 'a' from state A, it goes to state B)

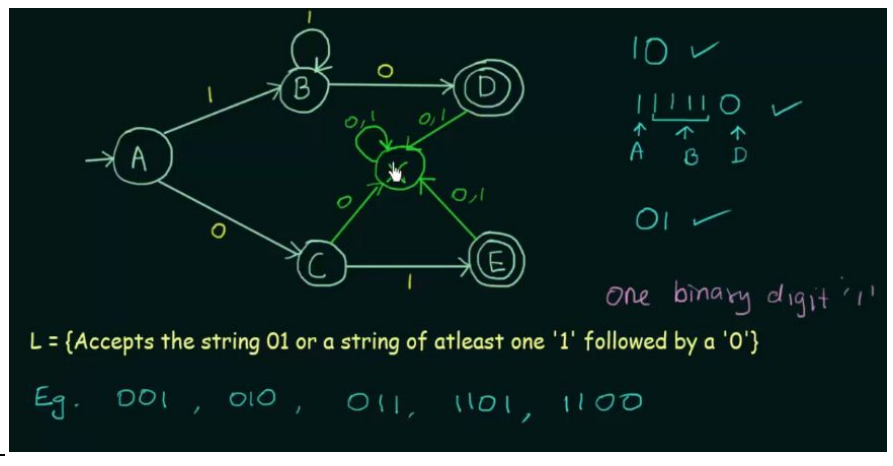
NOW WE NEED TO REVERSE ……..

1) All final state becomes normal state
2) All normal state becomes final state

L3=



EXAMPLE 4:



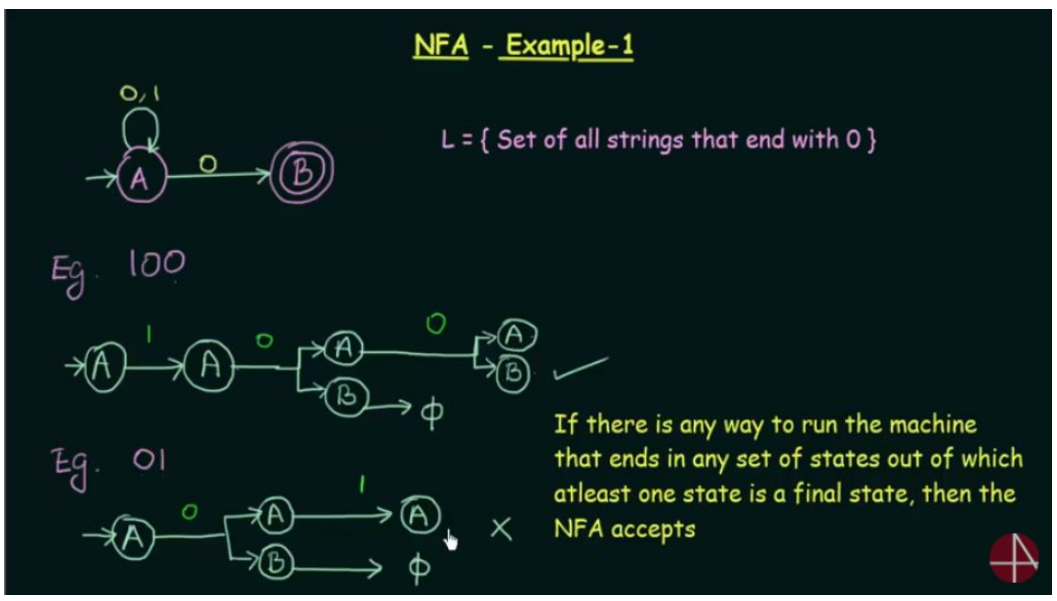
Non deterministic Finite automata:

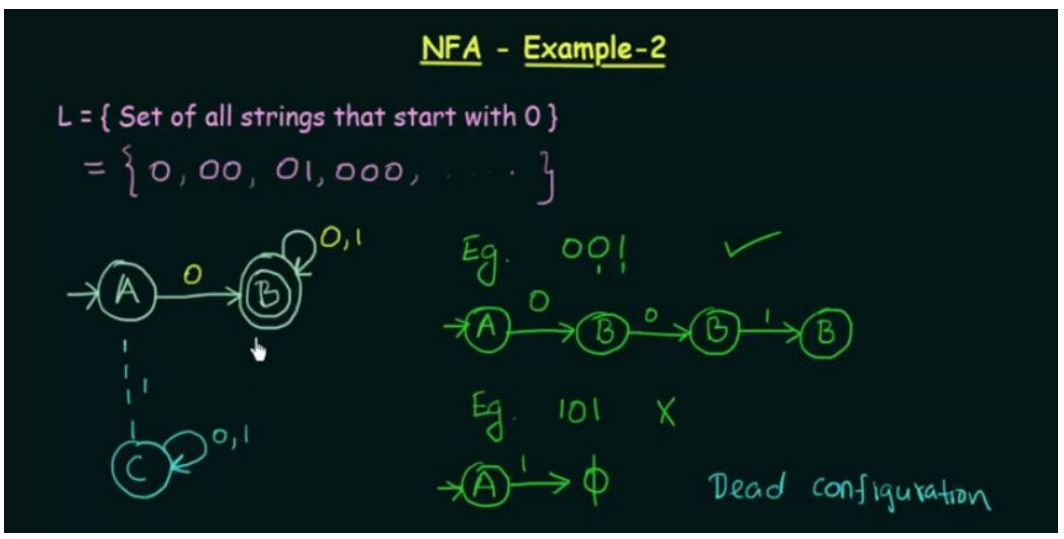
-Given the current state there could be many next states
-The next state might be chosen at random

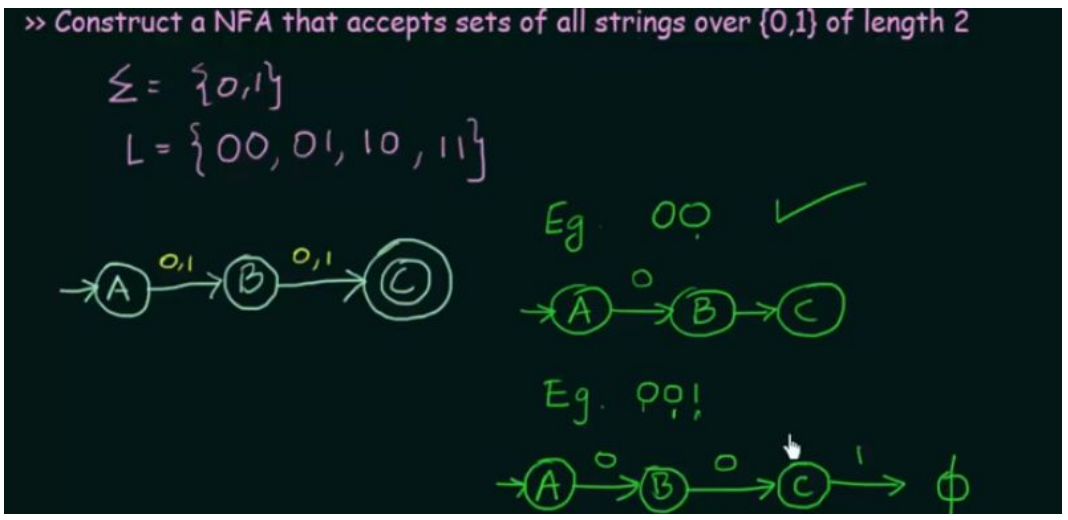
OR...

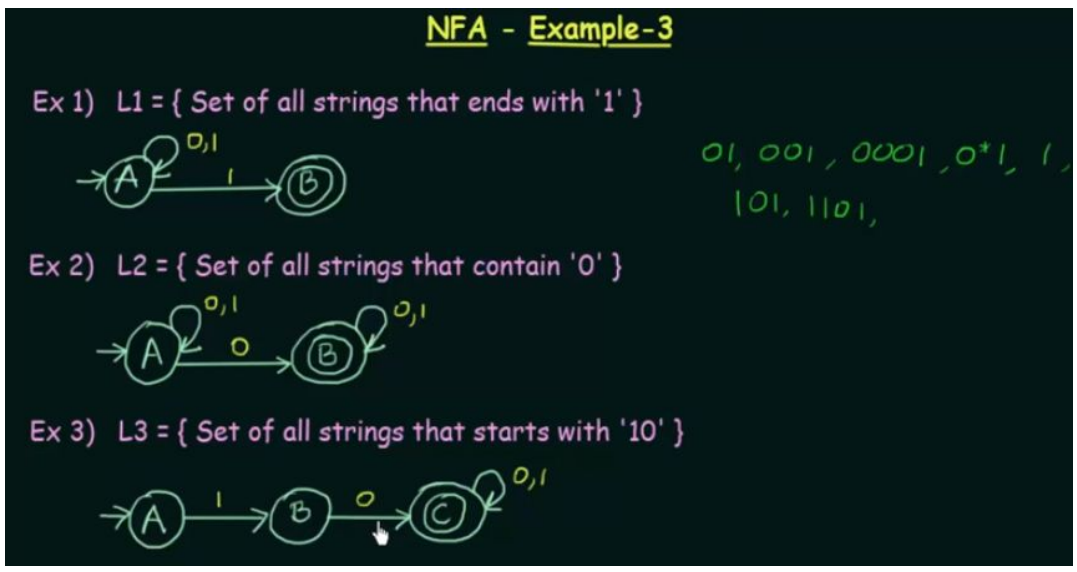
-All the next states may be chosen in parallel
-empty string " ε" could also be an input

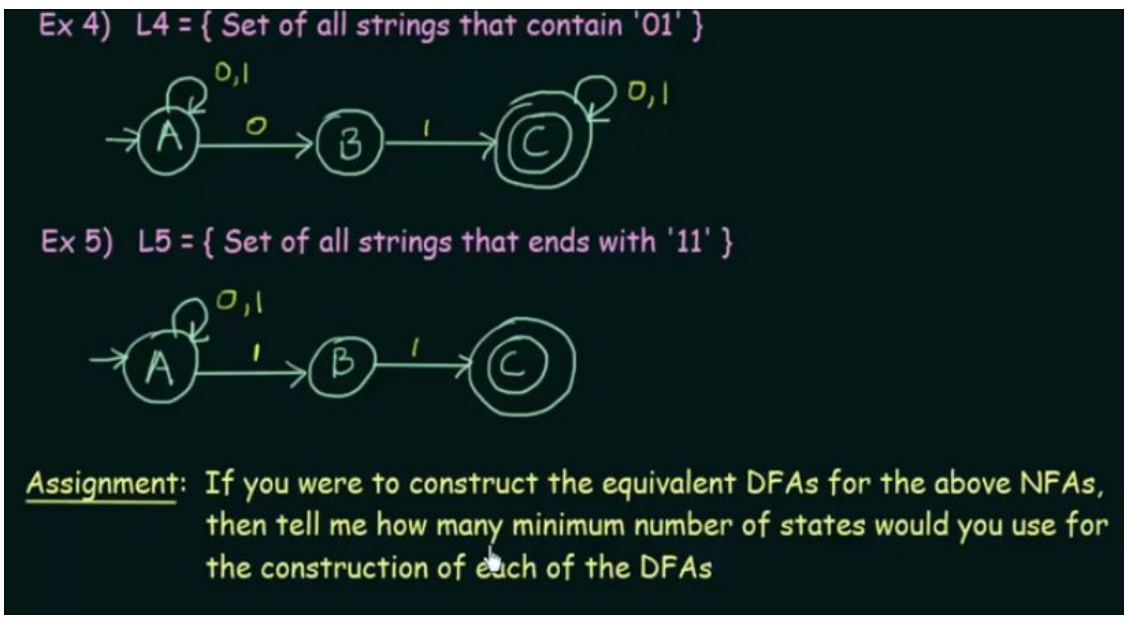
For NFA:
Q = set of all states
Q0 = initial state
F = set of final states
Sigma = inputs
Dell = Q x sigma ---> 2 ^Q







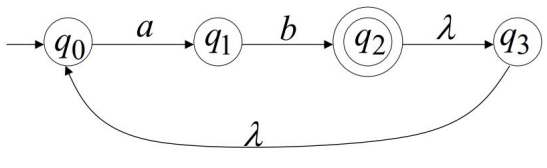




1)2states
2)2states
3)4states KEYWORD: THAT **STARTS: IF IT STARTS OTHERWISE, IT GOES TO DEAD STATE**
4)3states KEYWORD: **CONTAINS : IF IT MESSES UP, CAN ALWAYS GO BACK TO FIRST POSITION**
5)3states KEYWORD: **ENDS : IF IT MESSES UP, CAN ALWAYS GO BACK TO POSITION**

Language accepted

$$L = \{ab,\ abab,\ ababab,\ ...\}$$
$$= \{ab\}^+$$

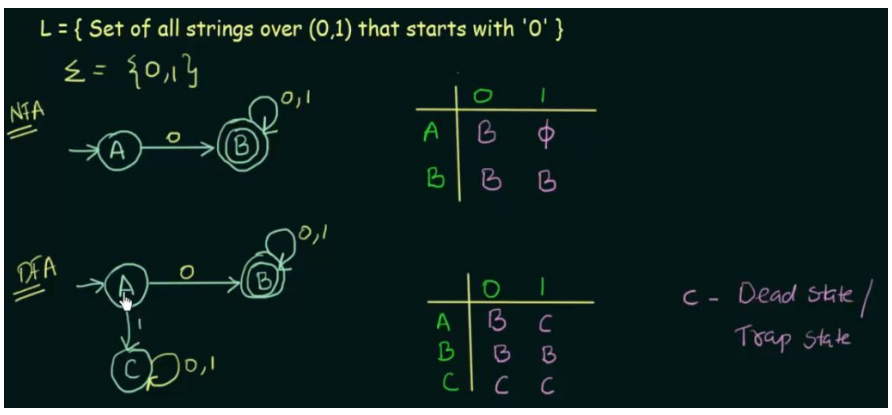


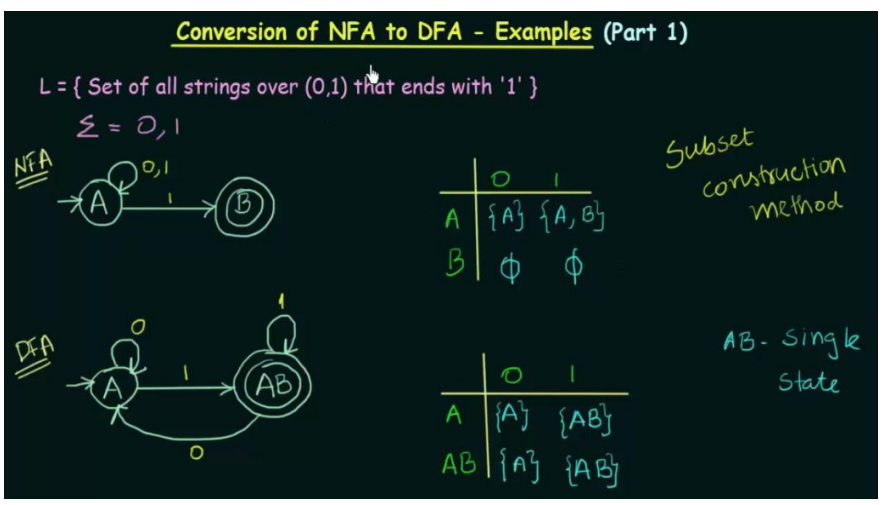
CONVERSION NFA TO DFA:

**EVERY DFA IS NFA

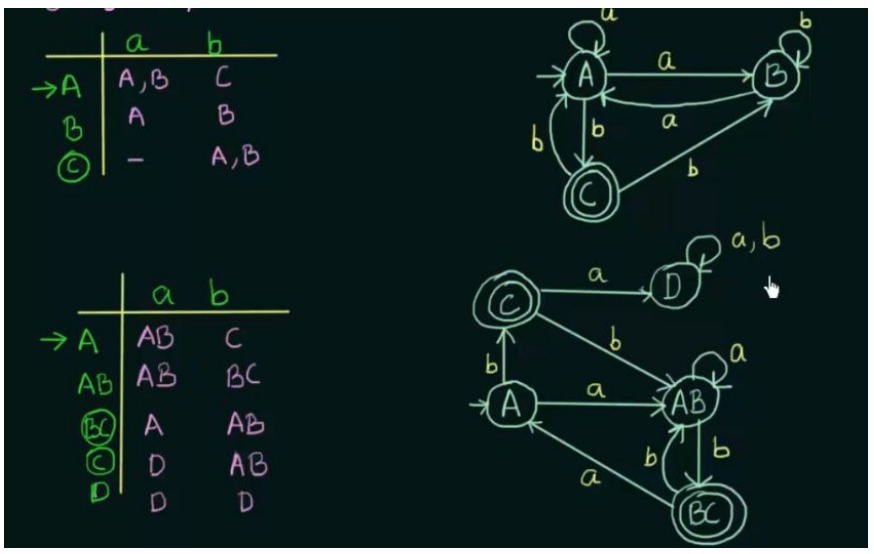
BUT NOT ALL NFA ARE DFA.

HOWEVER, THERE IS AN EQUIVALENT DFA FOR EVERY NFA



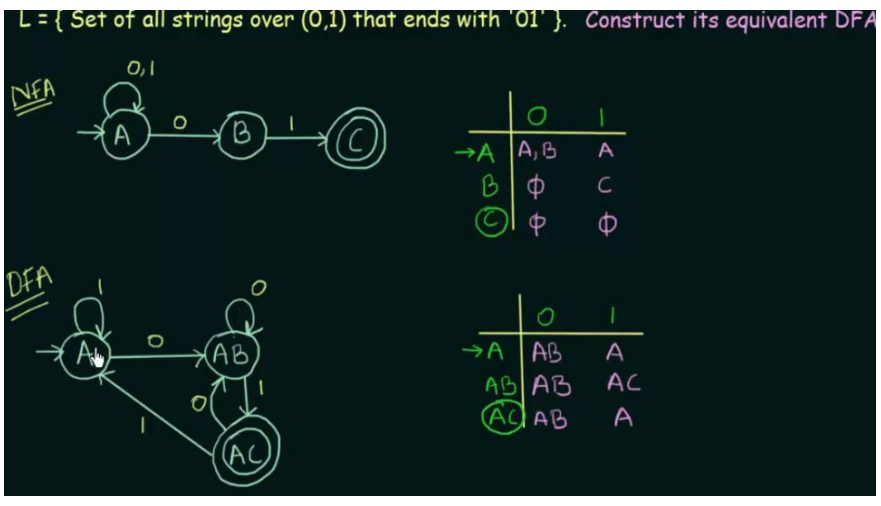


EXAMPLE 2:

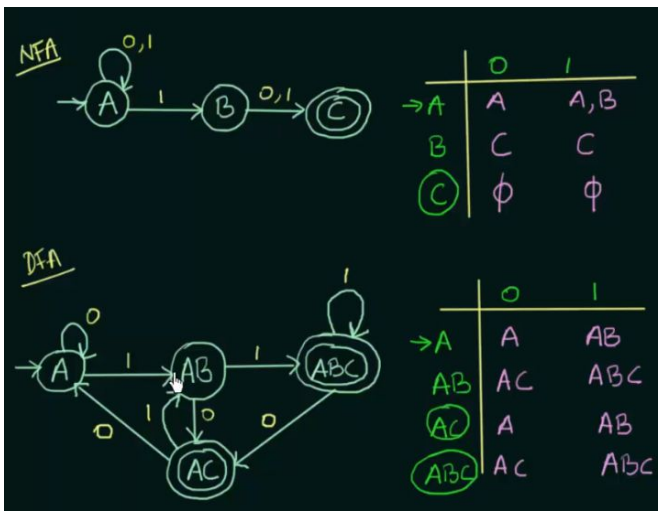


The DFA and NFA above represent the language L that accepts all strings that end with an odd number of ‘b’ and that must start with ‘ab’.

EXAMPLE 3:

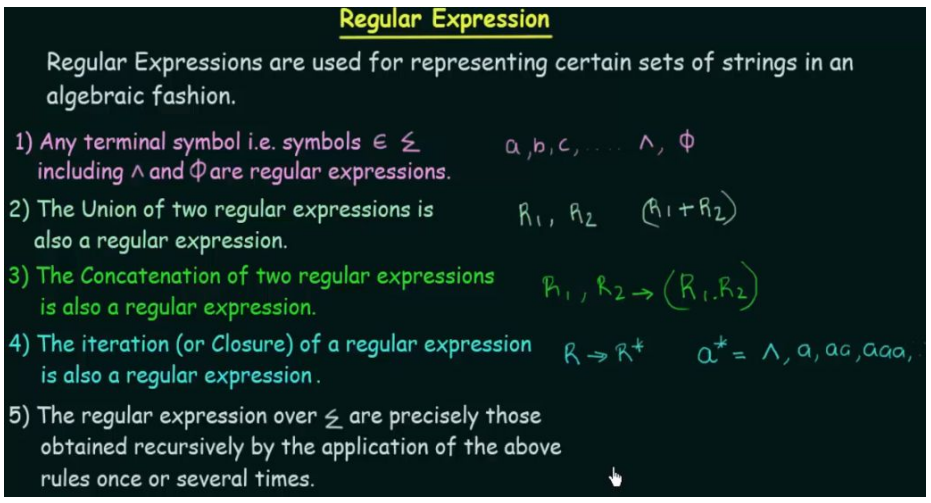


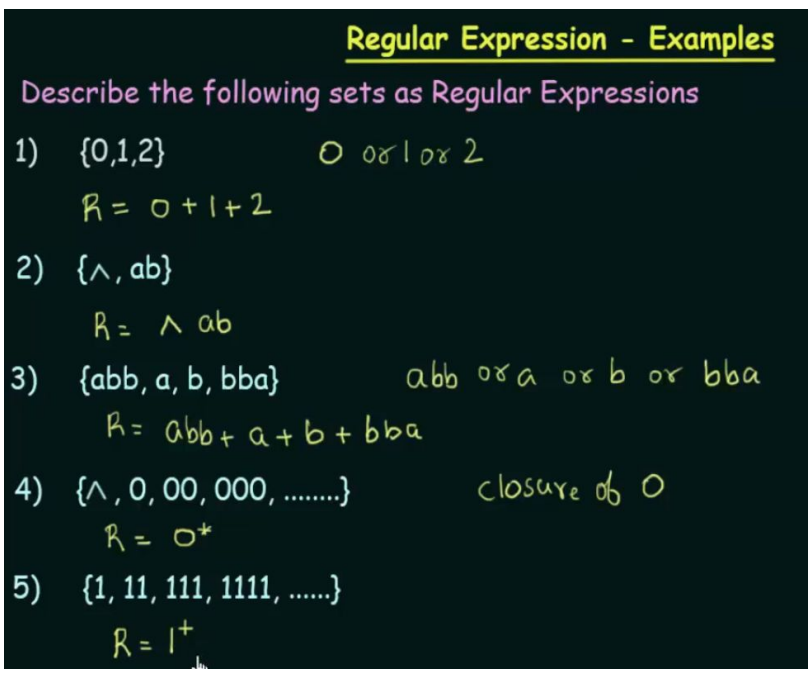
EXAMPLE 4: CREATE AN NFA THAT ACCEPTS ALL STRINGS WHICH HAS ‘1’ AS SECOND LAST

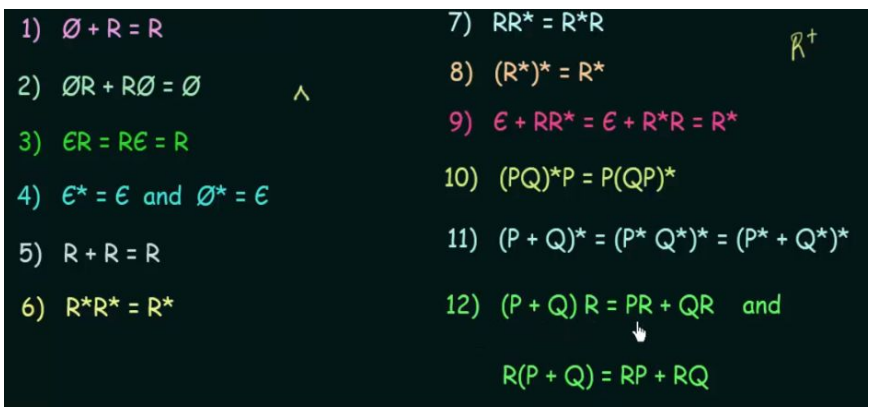


**IF THE NFA ACCEPTS EMPTY STRINGS, THEN INITIAL STATE IN THE DFA IS AUTOMATICALLY A FINAL STATE

REGULAR EXPRESSIONS:

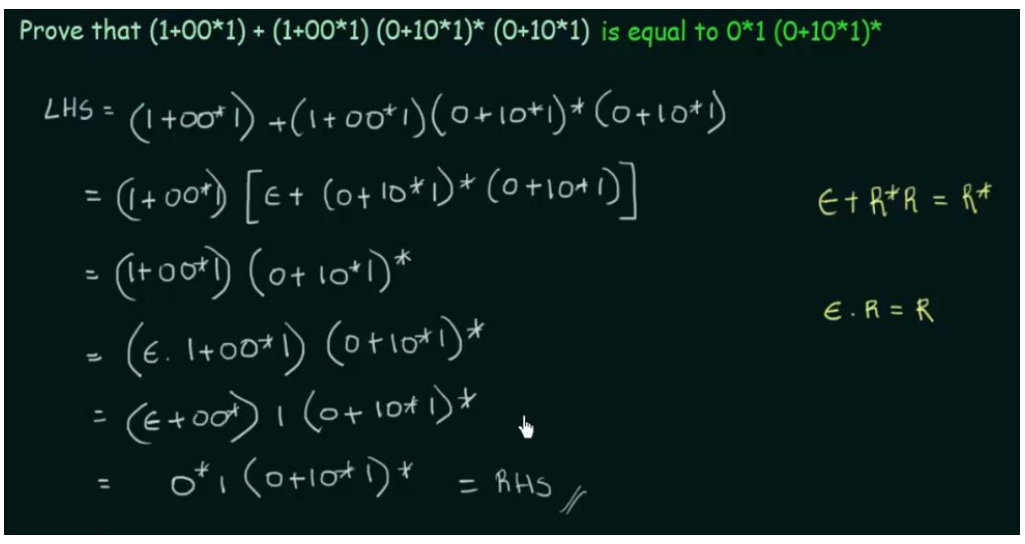






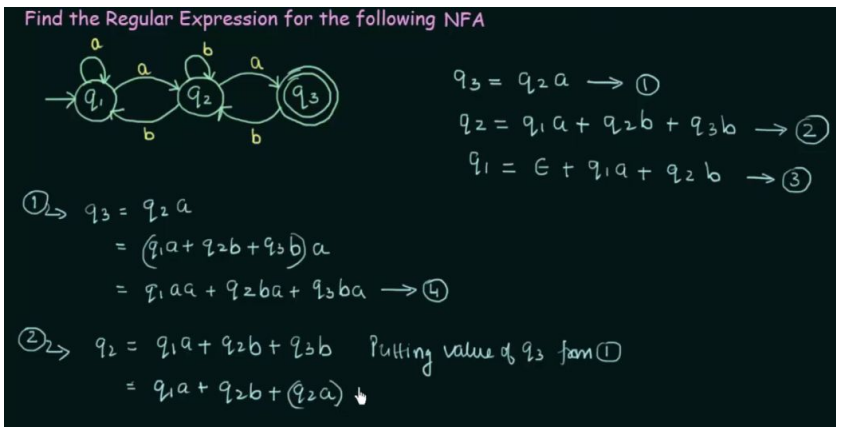
ARDEN'S THEOREM : IF R=Q+RP ---->R = PQ*

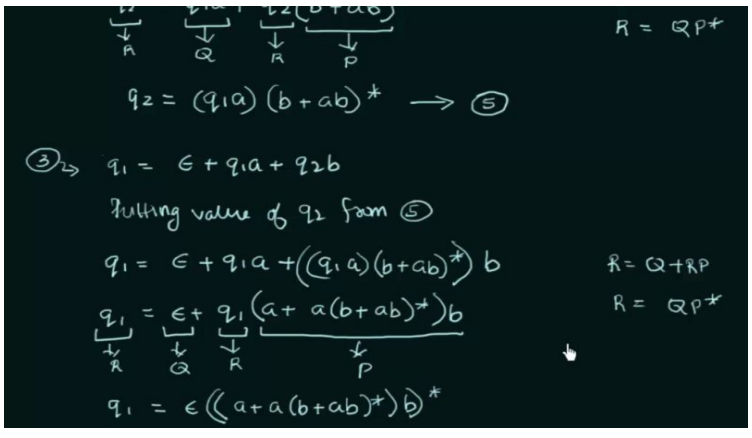
EXAMPLE PROOF 1 :

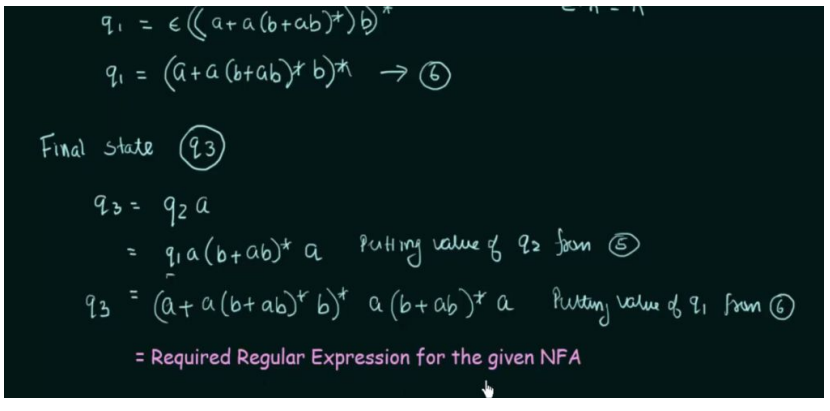


CONVERSION OF NFA/DFA TO REGULAR EXPRESSION:

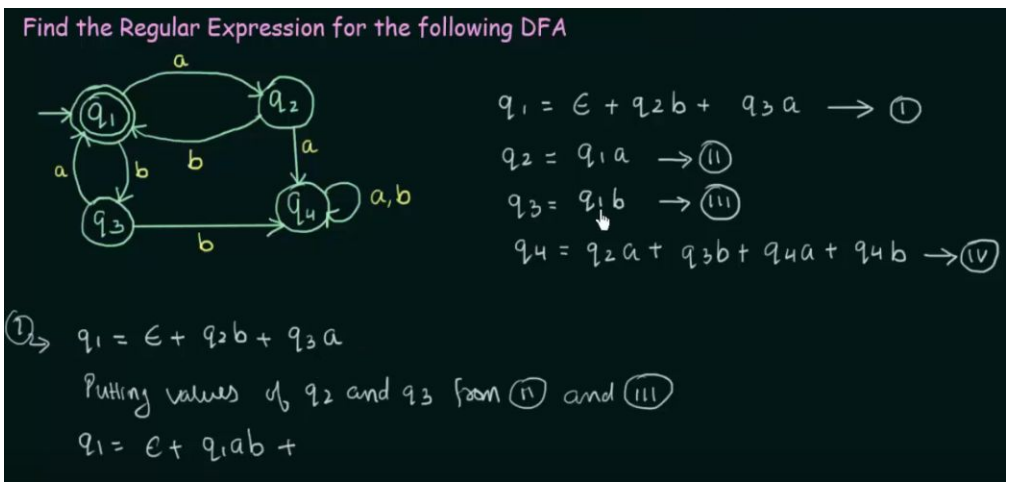
EXAMPLE 2: CONVERTING NFA TO REGULAR EXPRESSION

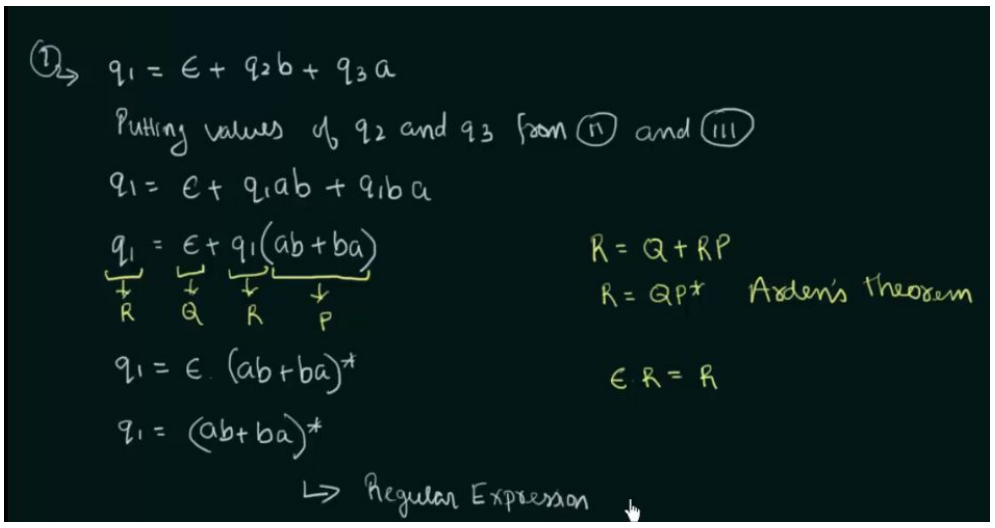




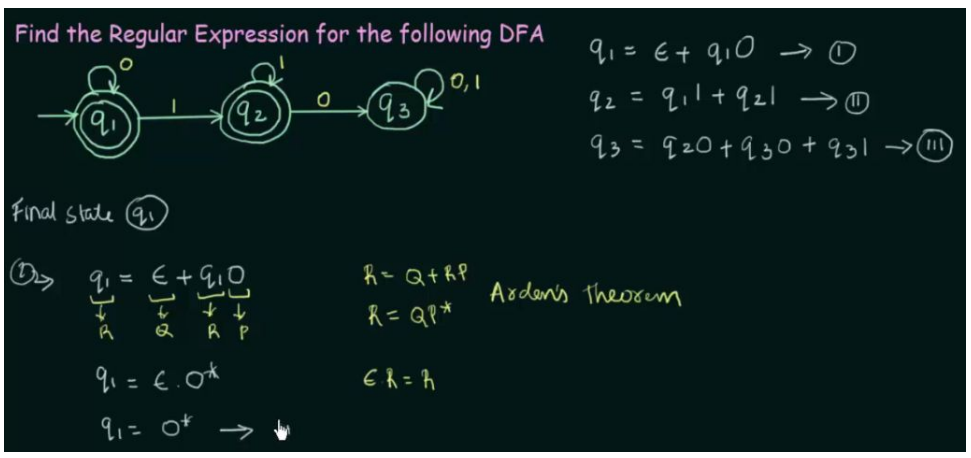


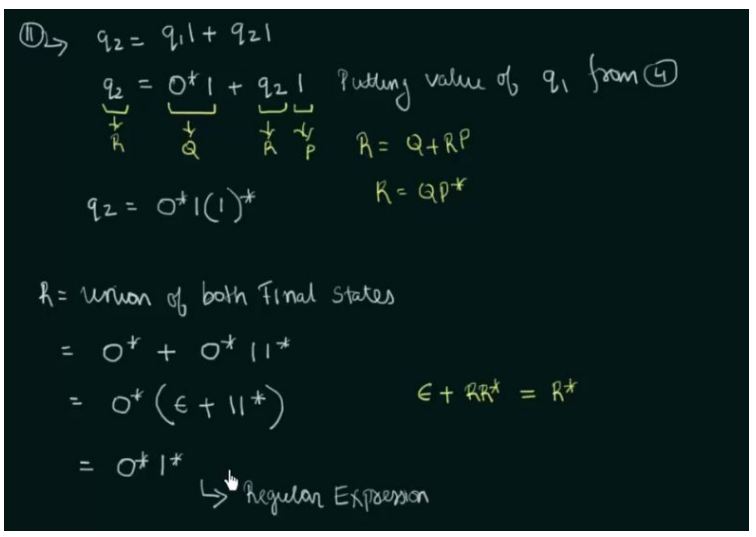
EXAMPLE 3: DFA TO REGULAR EXPRESSION



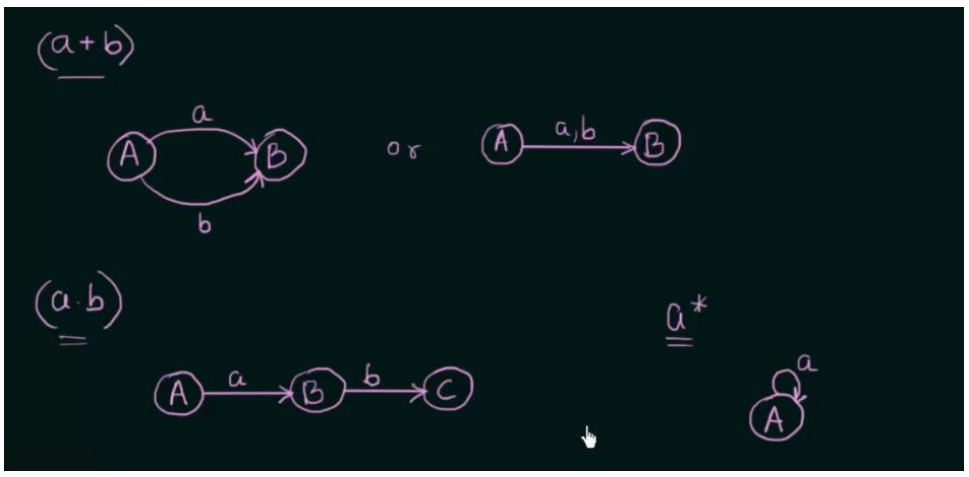


EXAMPLE 4: DFA WITH MULTIPLE FINAL STATES TO REGULAR EXPRESSION:

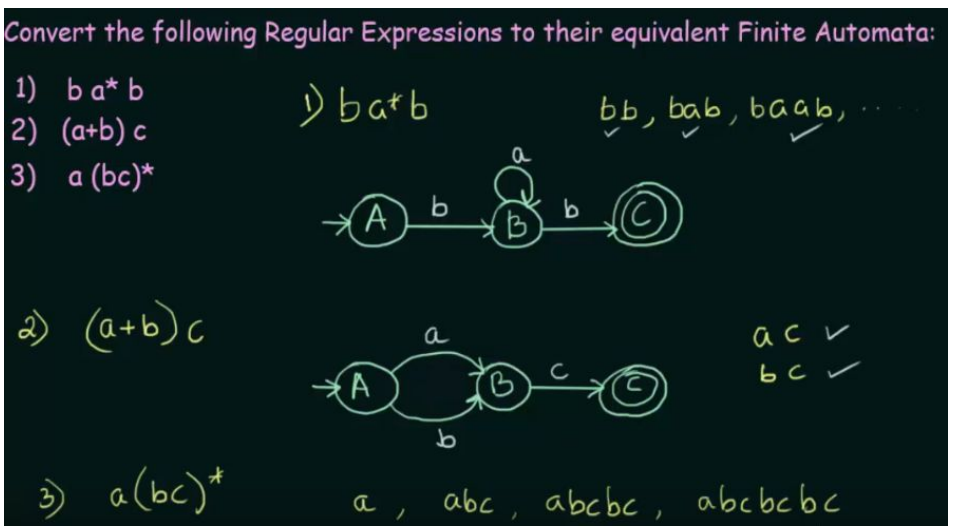


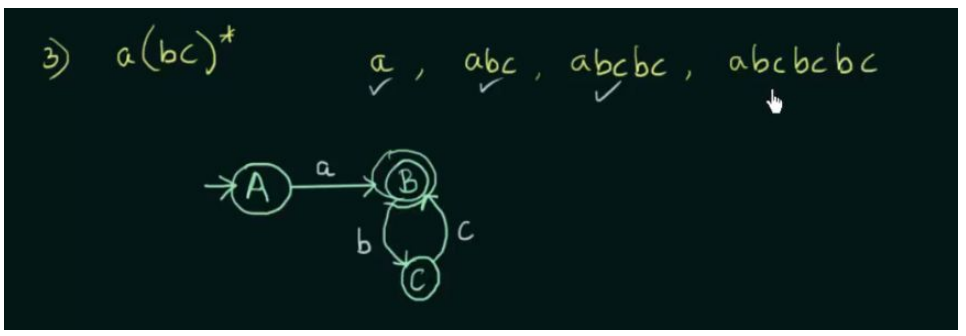


REGULAR EXPRESSION TO FINITE AUTOMATA:

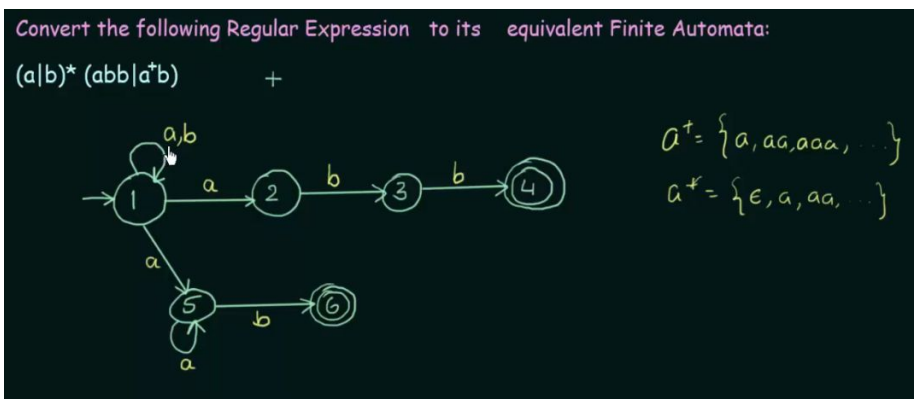


Example 1:





Example 2:

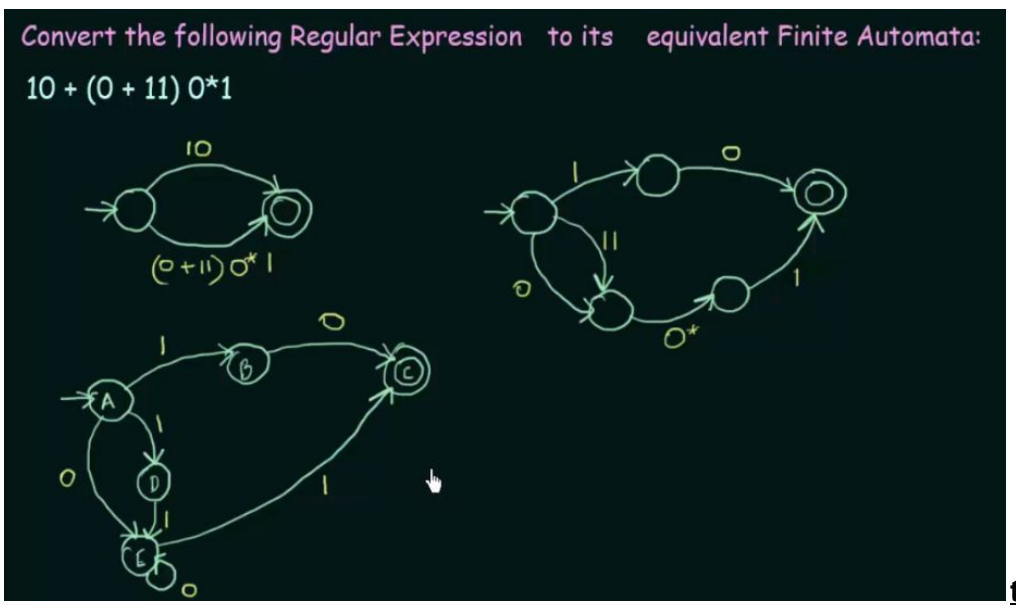


1) '|' means or. +

2) (a|b)* means recursion with possible empty string, while a b means recursion with no empty string. This means there has to be at least 1 'a'.

3) There are 2 finite states. Both ends with 'b'

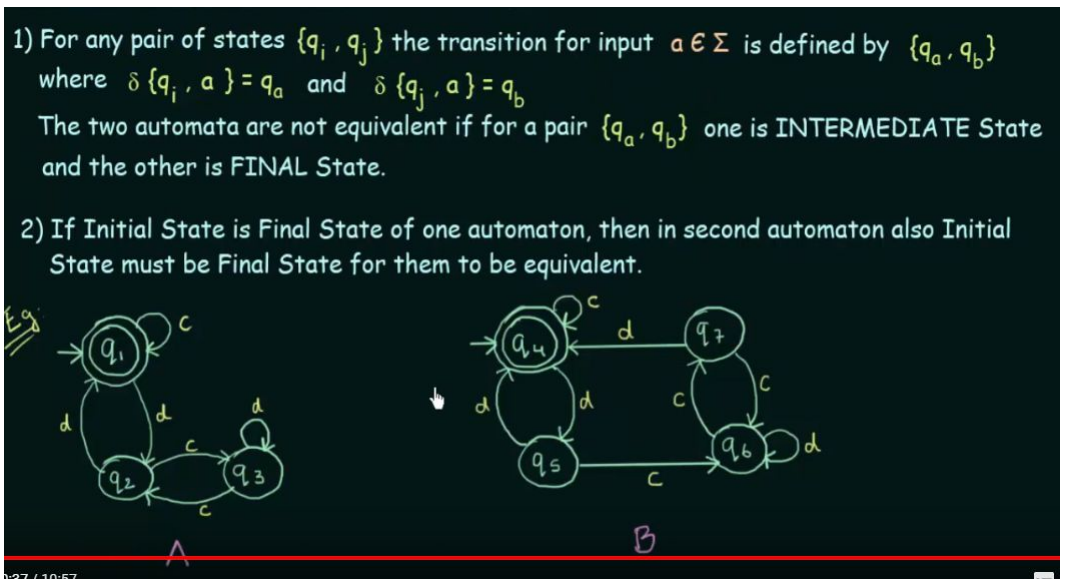
EXAMPLE 3:

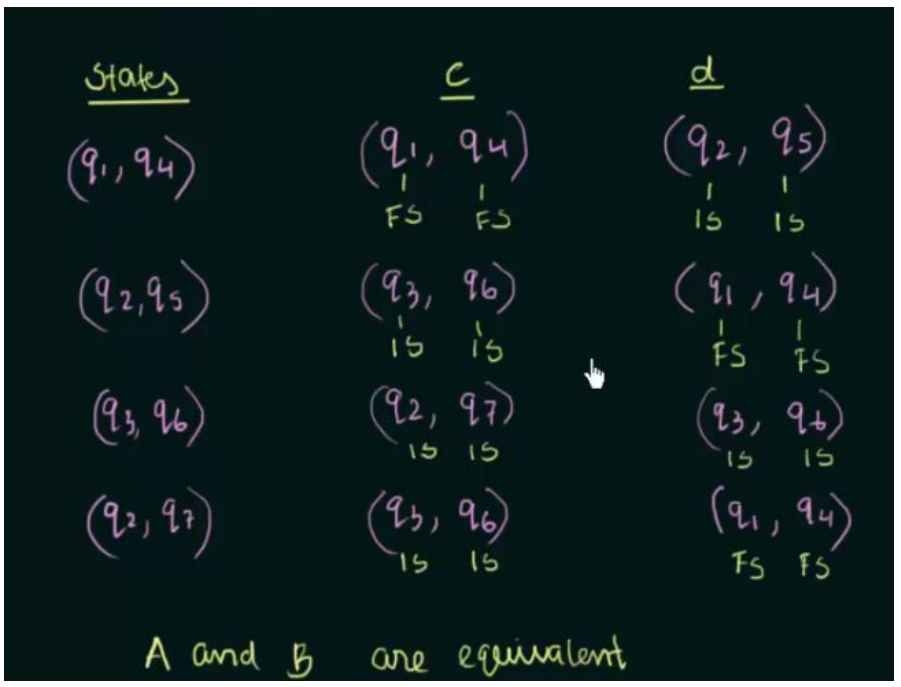


try to do i step by step

EQUIVALENT FINITE AUTOMATAS :

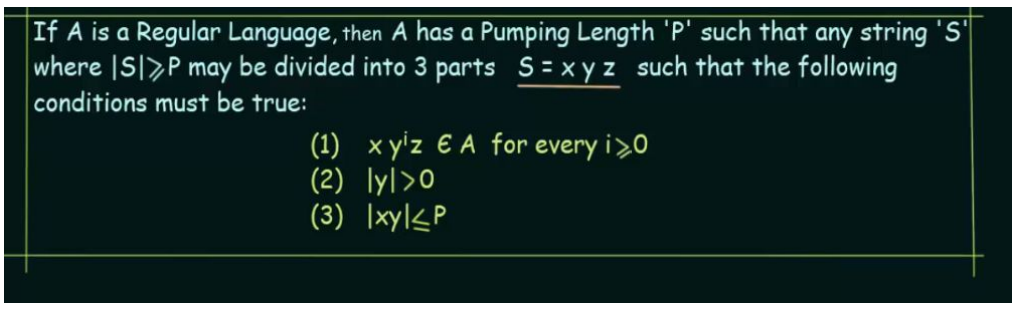
Example 1:

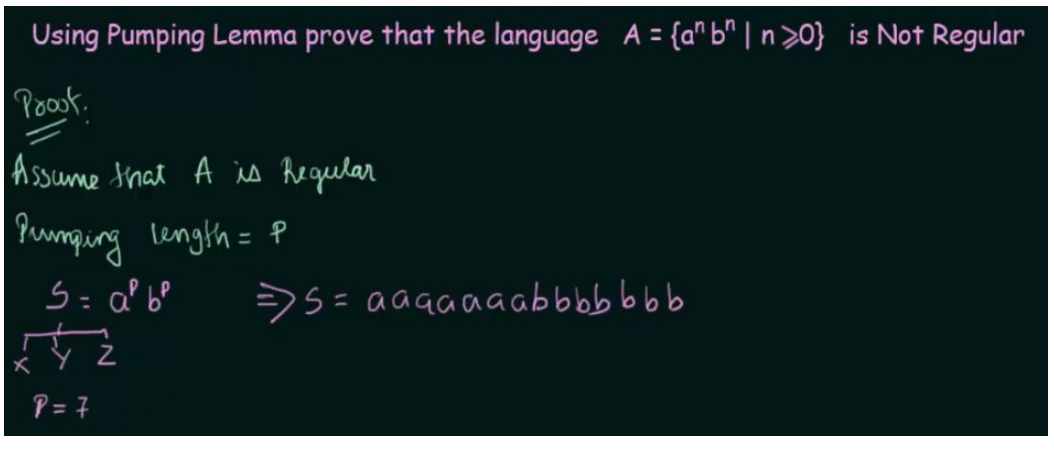




Example 2:

PUMPING LEMMA FOR REGULAR LANGUAGES





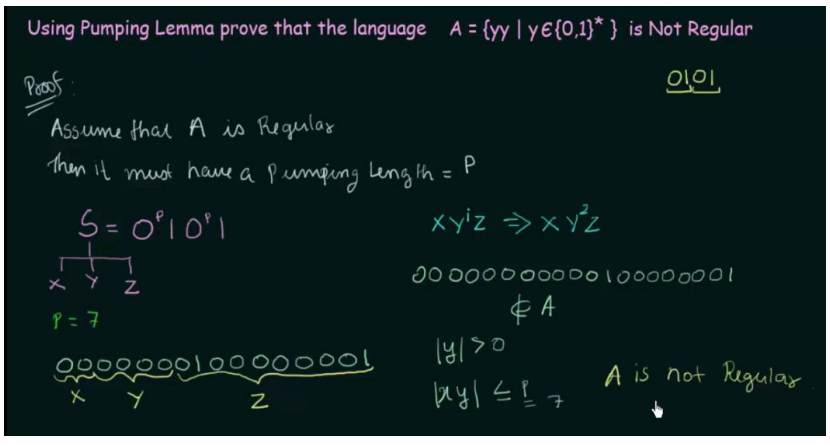
STEP 1: ASSUME A LENGTH FOR PUMPING LENGTH ‘P’. IN THIS CASE, P= 7

STEP 2: CONSTRUCT THE STRING S BASED OFF P

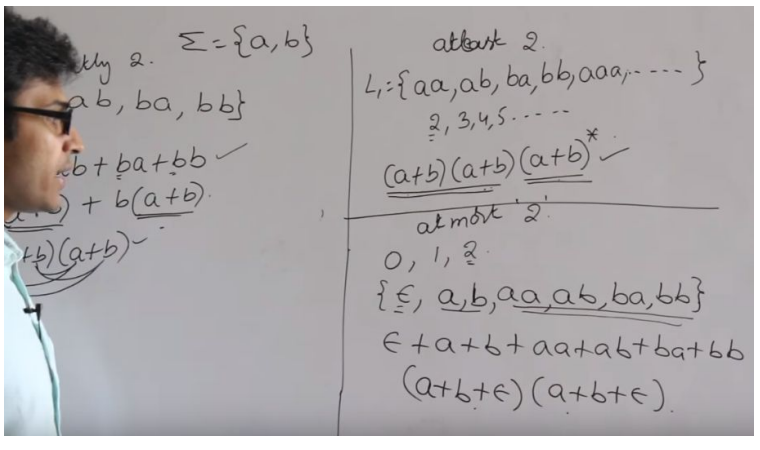
STEP 3: DIVIDE THE STRING S INTO 3 PARTS, X AND Y AND Z. CONSIDER EACH CAS. IN THIS EXAMPLE, Y COULD BE IN ‘a’, ‘b’ or both.

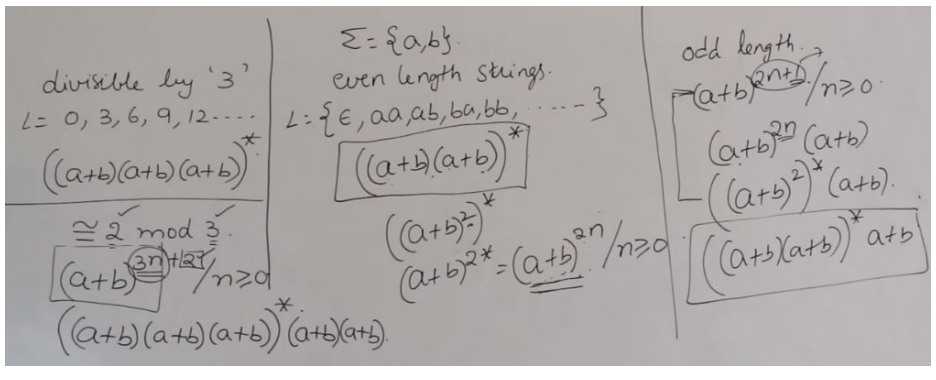
STEP 4:CAREFUL!!! |XY| HAS TO BE >= P

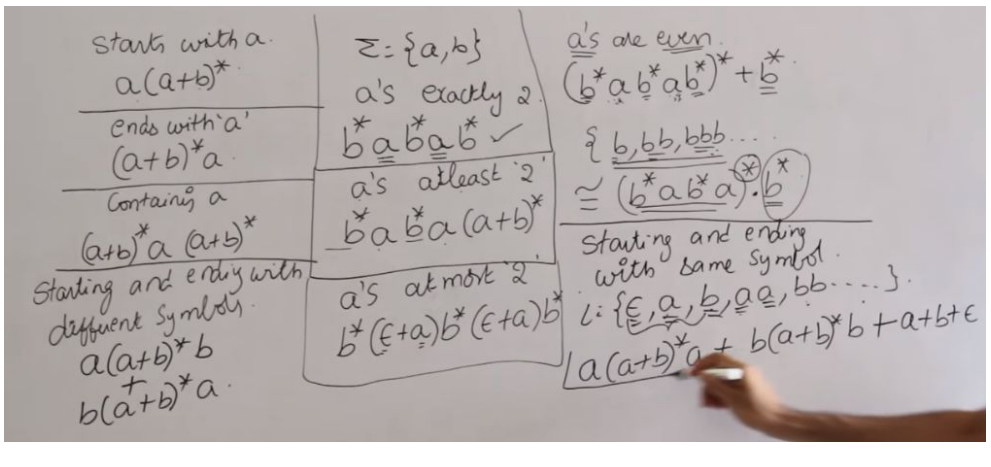
STEP 5: FOR SOME ‘i’, show that xy^iz **is not PART OF THE REGULAR LANGUAGE**

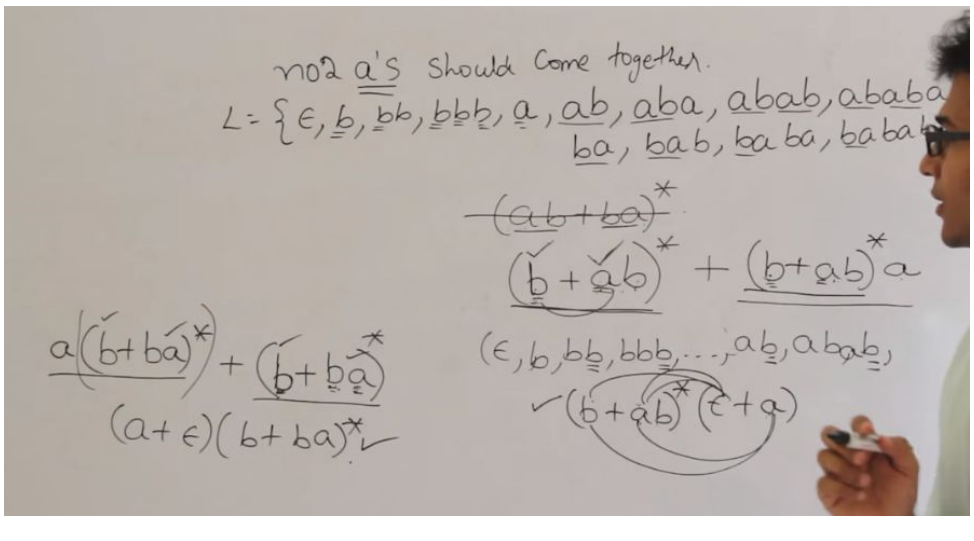


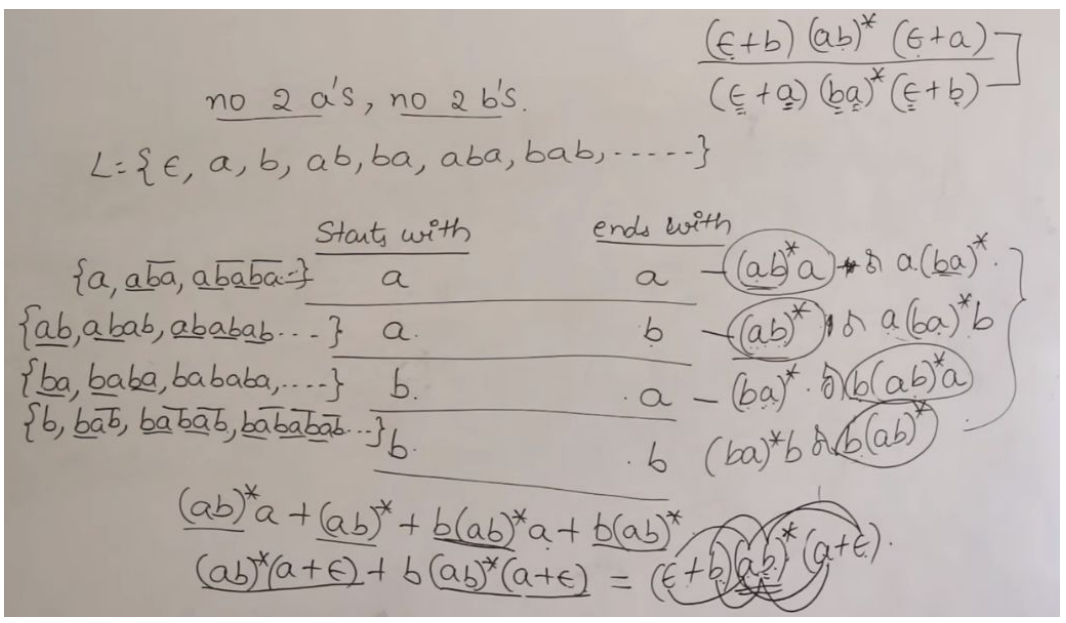
EXAMPLE OF REGULAR EXPRESSIONS !!

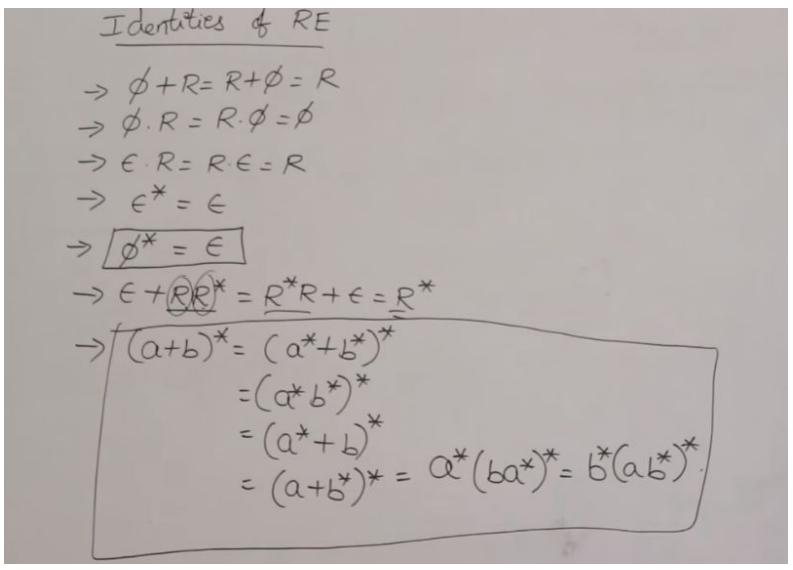




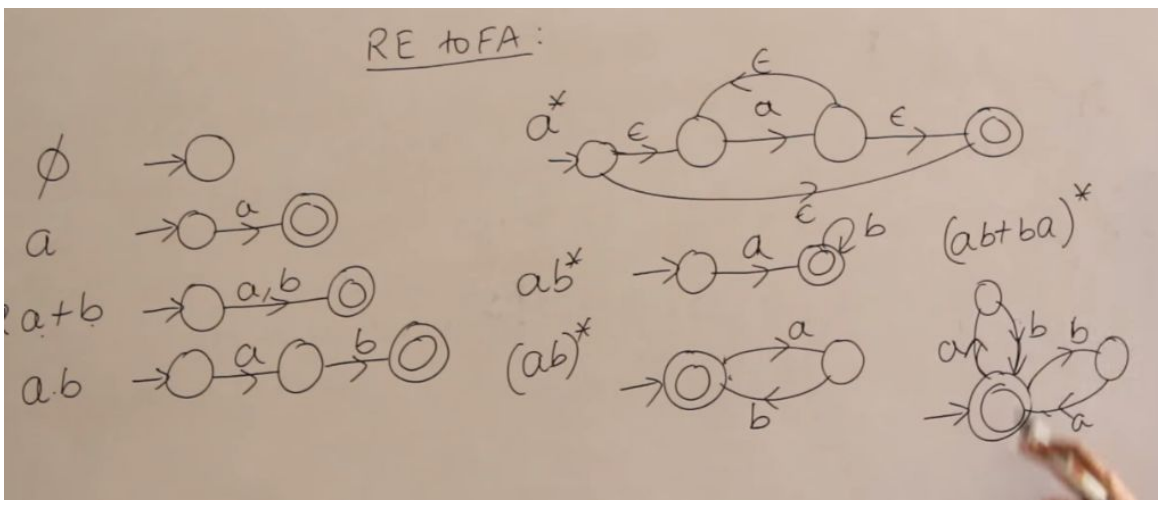








Conversion REG TO FA:



CONVERSION FA TO REG:

STATE ELIMINATION METHOD:

1) There should be no incoming input for starting state----->create new initial state with epsilon transition hooked up from new to old initial state
2) There should be no outgoing input for ending state------->create new final state with epsilon transition linking from old to new final state
3) There should be only 1 ending state
4) After applying the changes, eliminate any non starting/ending states

GRAMMAR:

Grammar:

A Grammar 'G' can be formally described using 4 tuples as G = (V,T,S,P) where,

V = Set of Variables or Non-Terminal Symbols
T = Set of Terminal Symbols
S = Start Symbol
P = Production rules for Terminals and Non-Terminals

A production rules has the form $\alpha \longrightarrow \beta$ where α and β are strings on V U T and atleast one symbol of α belongs to V.

Example: G = ({S,A,B}, {a,b}, S, {S→AB, A→a, B→b})

V = {S, A, B}
T = {a, b}
S = S
P = S→AB, A→a, B→b

Eg: S → AB → aB → ab

Regular Grammar:

Regular Grammar can be divided into two types:

Right Linear Grammar	Left Linear Grammar
A grammar is said to be Right Linear if all productions are of the form A→xB A→x where A,B ∈ V and x ∈ T	A grammar is said to be Left Linear if all productions are of the form A→Bx A→x where A,B ∈ V and x ∈ T

Eg: S → abS | b - Right linear

S → Sbb | b - Left linear

DERIVATION FROM A GRAMMAR:

The set of all strings that can be derived from a Grammar is said to be the LANGUAGE generated from that Grammar

Example 1: Consider the Grammar G1 = ({S,A}, {a,b}, S, {S→ aAb, aA→ aaAb, A→ ∈ })

S → aAb [by S → aAb]
→ aaAbb [by aA → aaAb]
→ a aaAbbb [by aA → aaAb]
→ aaabbb [by A → ∈]

Example 2: G2 = ({S,A,B}, {a,b}, S, {S→AB, A→a, B→b})

S → AB
→ ab

L(G2) = {ab}

Example 3: G3 = ({S,A,B}, {a,b}, S, {S→AB, A→aA|a, B→bB|b})

S → AB → ab

S → AB → aAbB → aabb

S → AB → aAb → aab

S → AB → abB → abb

L(G3) = {ab, a²b², a²b, ab², ...}
= {aᵐbⁿ | m ≥ 0 and n ≥ 0}

HOW TO FIND IF A STRING BELONGS TO A GRAMMAR:

1) Start with the Start Symbol and choose the closest production that matches to the given string.
2) Replace the Variables with its most appropriate production. Repeat the process until the string is generated or until no other productions are left.

Example: Verify whether the Grammar S→0B|1A, A→0|0S|1AA|^, B→1|1S|0BB generates the string 00110101

S → 0B (S → 0B)
→ 00BB (B → 0BB)
→ 001B (B → 1)
→ 0011S (B → 1S)
→ 00110B (S → 0B)
→ 001101S (B → 1S)
→ 0011010B (S → 0B)
→ 00110101 (B → 1)

Example: Verify whether the Grammar S→aAb, A→aAb|^ generates the string aabbb

S → aAb
→ a aAbb (A → aAb)
→ aabb (A → ^)
aa aAbbb (A → aAb)
aaabbb (A → ^) ✗

^ MEANS EMPTY STRING