

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: $A^* = \{x_1 x_2 x_3 x_4 \dots x_k \mid k \geq 0, x_k \text{ is element of } A\}$

- 1) Union : $A \cup B = \{pq, r, t, uv\}$
- 2) Concatenation : $A \circ B = \{pqt, pquv, rt, ruv\}$
- 3) Star: $A^* = \{\epsilon, pq, r, pqr, rpq, pqpq, rr, pqpqpq, rrr, \dots\}$

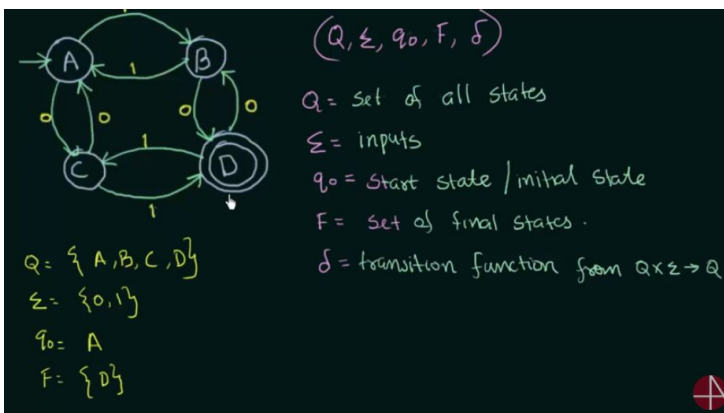
Theorem 1: The union of 2 regular languages is also a regular language

Theorem 2: The concatenation of 2 regular languages is also a regular language

PREREQUISITES:

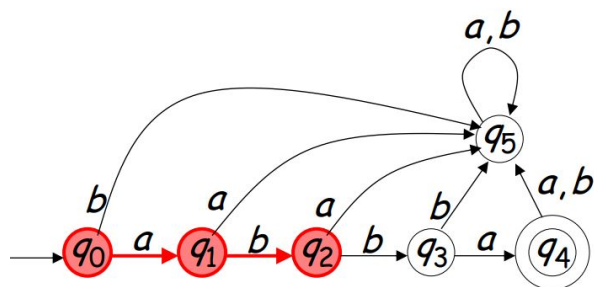
- Symbols
- Alphabet : denoted by Σ , collection of symbols. Ex : $\{a,b\}, \{0,1,2,3,\dots\}$
- String: Sequence of symbols . Ex: a,b,b,a
- Language: Set of strings / Ex: $L_1 = \text{Set of all strings of length 2 for } \Sigma = \{0,1\} \rightarrow L_1 = \{00,01,10,11\}$
- Powers of Σ : Ex: $\Sigma^0 = \text{set of strings of length 0} = \{\epsilon\}$, $\Sigma^2 = \text{set of strings of length 2} = \{00,01,10,11\}$...
- Cardinality: number of elements in a set. Ex: Cardinality of Σ^2 is 4. \rightarrow Cardinality = 2^n for alphabet of 2 symbols.
- Σ^* = set of all possible strings of all length over $\{0,1\}$ \rightarrow it's infinite

DFA:



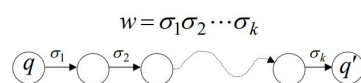
	0	1
A	C	B
B	D	A
C	A	D
D	B	C

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



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

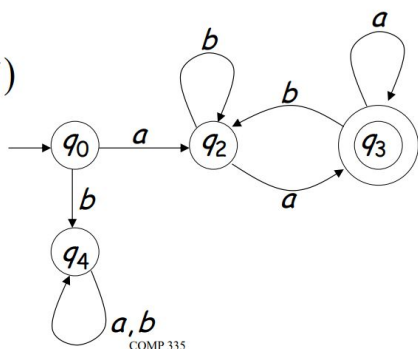
$$\delta^*(q, w) = q'$$



ANOTHER EXAMPLE

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

$$L = L(M)$$



w is a walk, which means it could be anything

NFA:

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

NFA - Formal Definition

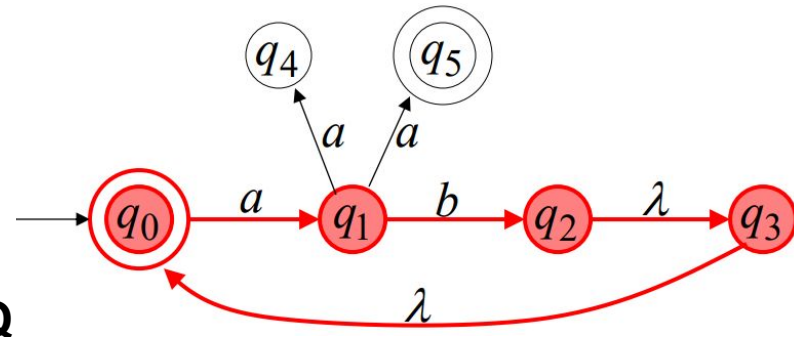
$L = \{ \text{Set of all strings that end with 0} \}$

$(Q, \Sigma, q_0, F, \delta)$

$Q = \text{Set of all states} \quad - \{A, B\}$
 $\Sigma = \text{inputs} \quad - \{0, 1\}$
 $q_0 = \text{start state / initial state} \quad - A$
 $F = \text{Set of final states} \quad - B$
 $\delta = Q \times \Sigma \rightarrow _ \quad - ?$

$A \times 0 \rightarrow A$
 $A \times 0 \rightarrow B$
 $A \times 1 \rightarrow A$
 $B \times 0 \rightarrow \phi$
 $B \times 1 \rightarrow \phi$

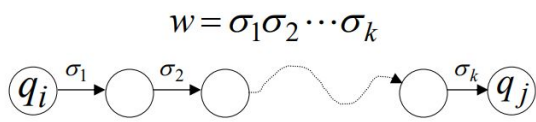
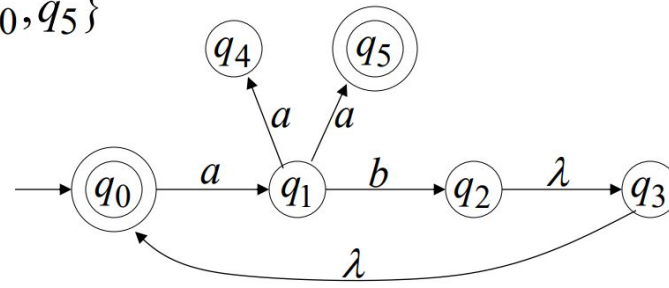
Del = 2^Q



Formally

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

$$F = \{q_0, q_5\}$$



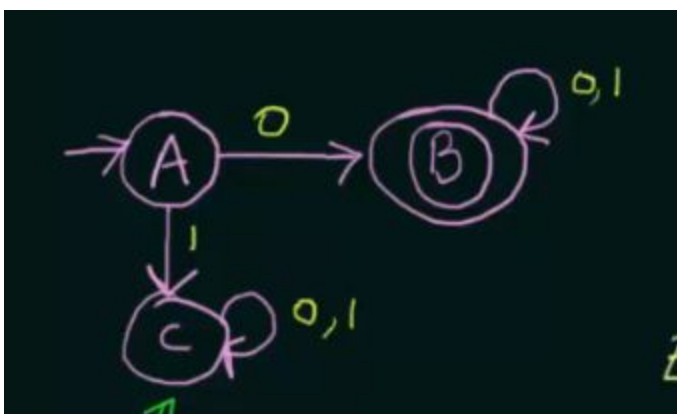
$$\delta^*(q_0, ab) = \{q_2, q_3, q_0\} \quad ab \in L(M)$$

 $\searrow \in F$

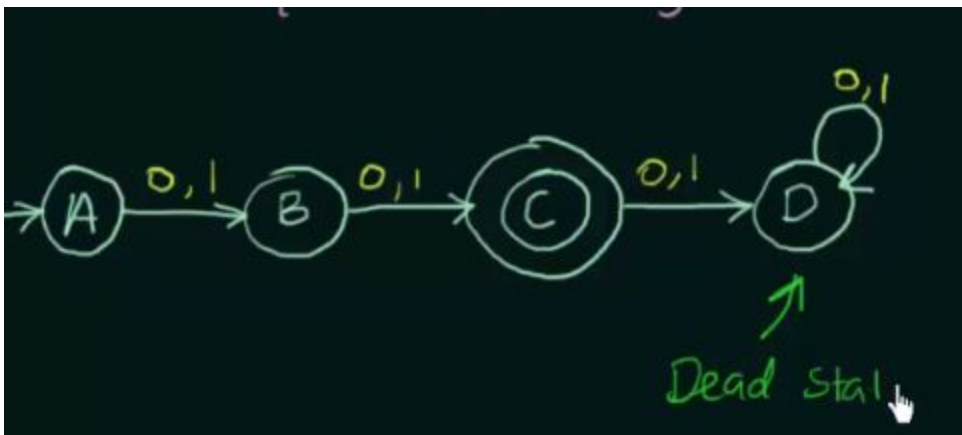
Deterministic Finite automata:

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

EXAMPLE 1: $L_1 = \text{SET OF ALL STRINGS THAT START WITH 0}$

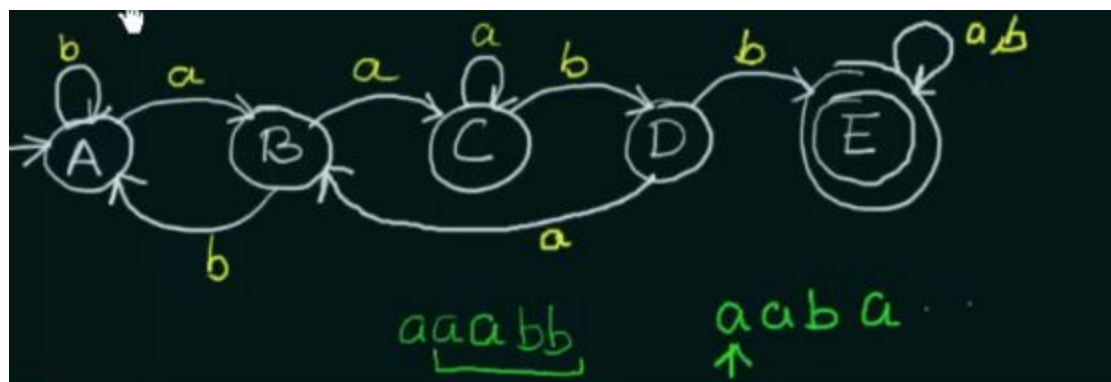


EXAMPLE 2: $L_2 = \text{SET OF ALL STRINGS OF LENGTH 2}$



EXAMPLE 3: $L_3 = \text{SET OF STRINGS OVER } \Sigma=\{a,b\} \text{ THAT DOES NOT CONTAIN THE STRING } aabb \text{ IN IT.}$

$\Rightarrow \text{WRITE } L_3^* = \text{SET OF ALL STRINGS THAT CONTAIN } aabb \text{ !!!}$

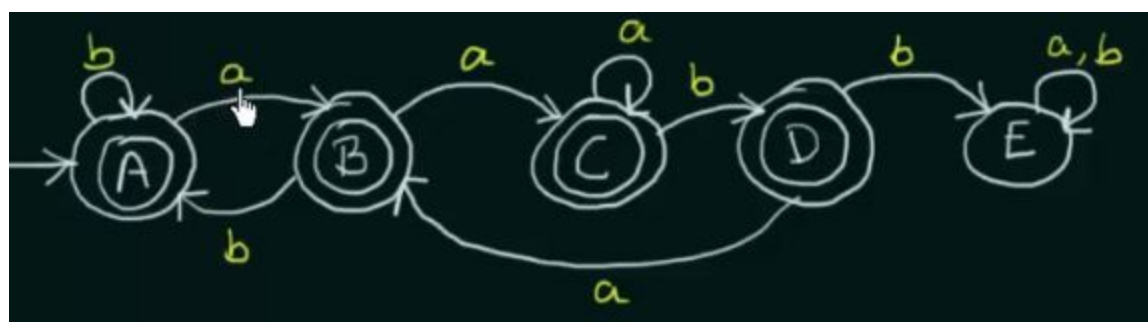


$L_3^* =$

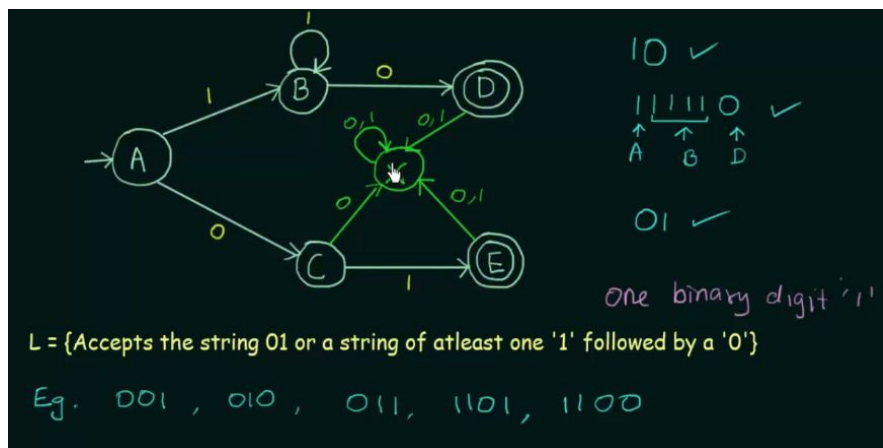
from state D, if input is 'a' then it goes back to the position of the first 'a', which is @ state B (upon first 'a' from state A, it goes to state B)

NOW WE NEED TO REVERSE

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



$L_3 =$



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

NFA - Example-1

L = { Set of all strings that end with 0 }

Eg. 100

Eg. 01

If there is any way to run the machine that ends in any set of states out of which atleast one state is a final state, then the NFA accepts

NFA - Example-2

L = { Set of all strings that start with 0 }

= { 0, 00, 01, 000, ... }

Eg. 001 ✓

Eg. 101 X

Dead configuration

>> Construct a NFA that accepts sets of all strings over {0,1} of length 2

$\Sigma = \{0,1\}$

$L = \{00, 01, 10, 11\}$

Eg. 00 ✓

Eg. 001

NFA - Example-3

Ex 1) L1 = { Set of all strings that ends with '1' }

01, 001, 0001, 0*1, 1, 101, 1101,

Ex 2) L2 = { Set of all strings that contain '0' }

Ex 3) L3 = { Set of all strings that starts with '10' }

Ex 4) L4 = { Set of all strings that contain '01' }

Ex 5) L5 = { Set of all strings that ends with '11' }

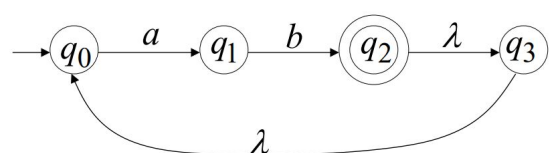
Assignment: If you were to construct the equivalent DFAs for the above NFAs, then tell me how many minimum number of states would you use for the construction of each of the DFAs

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

Language accepted

$$L = \{ab, abab, ababab, \dots\}$$

$$= \{ab\}^+$$



CONVERSION NFA TO DFA:

**EVERY DFA IS NFA

BUT NOT ALL NFA ARE DFA.

HOWEVER, THERE IS AN EQUIVALENT DFA FOR EVERY NFA

$L = \{ \text{Set of all strings over } (0,1) \text{ that starts with '0'} \}$
 $\Sigma = \{0,1\}$

NFA

	0	1
A	B	ϕ
B	B	B

DFA

	0	1
A	B	C
B	B	B
C	C	C

C - Dead state / Trap state

Conversion of NFA to DFA - Examples (Part 1)

$L = \{ \text{Set of all strings over } (0,1) \text{ that ends with '1'} \}$
 $\Sigma = 0,1$

NFA

	0	1
A	{A}	{A, B}
B	ϕ	ϕ

Subset construction method

DFA

	0	1
A	{A}	{AB}
AB	{A}	{AB}

AB - single state

EXAMPLE 2:

	a	b
\rightarrow A	A, B	C
B	A	B
C	-	A, B

	a	b
\rightarrow A	AB	C
AB	AB	BC
BC	A	AB
C	D	AB
D	D	D

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:

$L = \{ \text{Set of all strings over } (0,1) \text{ that ends with '01'} \}$. Construct its equivalent DFA

NFA

	0	1
\rightarrow A	A, B	A
B	ϕ	C
C	ϕ	ϕ

DFA

	0	1
\rightarrow A	AB	A
AB	AB	AC
AC	AB	A

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

NFA

	0	1
\rightarrow A	A	A, B
B	C	C
C	ϕ	ϕ

DFA

	0	1
\rightarrow A	A	AB
AB	AC	ABC
AC	A	AB
ABC	AC	ABC

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

REGULAR EXPRESSIONS:

Regular Expression

Regular Expressions are used for representing certain sets of strings in an algebraic fashion.

- 1) Any terminal symbol i.e. symbols $\in \Sigma$ including Λ and Φ are regular expressions. $a, b, c, \dots, \Lambda, \Phi$
- 2) The Union of two regular expressions is also a regular expression. $R_1, R_2 \rightarrow (R_1 + R_2)$
- 3) The Concatenation of two regular expressions is also a regular expression. $R_1, R_2 \rightarrow (R_1.R_2)$
- 4) The iteration (or Closure) of a regular expression is also a regular expression. $R \rightarrow R^+ \quad a^* = \Lambda, a, aa, aaa, \dots$
- 5) The regular expression over Σ are precisely those obtained recursively by the application of the above rules once or several times.

Regular Expression - Examples

Describe the following sets as Regular Expressions

- 1) $\{0,1,2\}$ $0 \text{ or } 1 \text{ or } 2$
 $R = 0 + 1 + 2$
- 2) $\{\Lambda, ab\}$
 $R = \Lambda + ab$
- 3) $\{abb, a, b, bba\}$ $abb \text{ or } a \text{ or } b \text{ or } bba$
 $R = abb + a + b + bba$
- 4) $\{\Lambda, 0, 00, 000, \dots\}$ closure of 0
 $R = 0^*$
- 5) $\{1, 11, 111, 1111, \dots\}$
 $R = 1^+$

- 1) $\emptyset + R = R$
- 2) $\emptyset R + R\emptyset = \emptyset$
- 3) $\epsilon R = R\epsilon = R$
- 4) $\epsilon^* = \epsilon$ and $\emptyset^* = \epsilon$
- 5) $R + R = R$
- 6) $R^*R^* = R^*$
- 7) $RR^* = R^*R$
- 8) $(R^*)^* = R^*$
- 9) $\epsilon + RR^* = \epsilon + R^*R = R^*$
- 10) $(PQ)^*P = P(QP)^*$
- 11) $(P + Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$
- 12) $(P + Q)R = PR + QR$ and
 $R(P + Q) = RP + RQ$

KLEIN'S THEOREM : IF $R = Q + RP \rightarrow R = PQ^*$

EXAMPLE PROOF 1 :

Prove that $(1+00^*1) + (1+00^*1)(0+10^*1)^*(0+10^*1)$ is equal to $0^*1(0+10^*1)^*$

$$\begin{aligned}
 \text{LHS} &= (1+00^*1) + (1+00^*1)(0+10^*1)^*(0+10^*1) \\
 &= (1+00^*1) [\epsilon + (0+10^*1)^*(0+10^*1)] && \epsilon + R^*R = R^* \\
 &= (1+00^*1)(0+10^*1)^* && \epsilon \cdot R = R \\
 &= (\epsilon + 00^*1)(0+10^*1)^* \\
 &= (\epsilon + 00^*1)1(0+10^*1)^* \\
 &= 0^*1(0+10^*1)^* = \text{RHS} //
 \end{aligned}$$

CONVERSION OF NFA/DFA TO REGULAR EXPRESSION:

EXAMPLE 2: CONVERTING NFA TO REGULAR EXPRESSION

Find the Regular Expression for the following NFA

$q_3 = q_2 a \rightarrow \textcircled{1}$
 $q_2 = q_1 a + q_2 b + q_3 b \rightarrow \textcircled{2}$
 $q_1 = \epsilon + q_1 a + q_2 b \rightarrow \textcircled{3}$

$\textcircled{1} \Rightarrow q_3 = q_2 a$
 $= (q_1 a + q_2 b + q_3 b) a$
 $= q_1 a a + q_2 b a + q_3 b a \rightarrow \textcircled{4}$

$\textcircled{2} \Rightarrow q_2 = q_1 a + q_2 b + q_3 b$ Putting value of q_3 from $\textcircled{1}$
 $= q_1 a + q_2 b + (q_2 a) b$

$R = QP^*$
 $q_2 = (q_1 a) (b + ab)^* \rightarrow \textcircled{5}$
 $\textcircled{3} \Rightarrow q_1 = \epsilon + q_1 a + q_2 b$
 Putting value of q_2 from $\textcircled{5}$
 $q_1 = \epsilon + q_1 a + ((q_1 a) (b + ab)^*) b$ $R = Q + RP$
 $R = QP^*$
 $q_1 = \epsilon + q_1 (a + a(b + ab)^* b)$
 $q_1 = \epsilon ((a + a(b + ab)^* b))^*$

$q_1 = \epsilon ((a + a(b + ab)^* b))^*$
 $q_1 = (a + a(b + ab)^* b)^* \rightarrow \textcircled{6}$

Final state q_3

$q_3 = q_2 a$
 $= q_1 a (b + ab)^* a$ Putting value of q_2 from $\textcircled{5}$
 $q_3 = (a + a(b + ab)^* b)^* a (b + ab)^* a$ Putting value of q_1 from $\textcircled{6}$
 $= \text{Required Regular Expression for the given NFA}$

EXAMPLE 3: DFA TO REGULAR EXPRESSION

Find the Regular Expression for the following DFA

$q_1 = \epsilon + q_2 b + q_3 a \rightarrow \textcircled{I}$
 $q_2 = q_1 a \rightarrow \textcircled{II}$
 $q_3 = q_1 b \rightarrow \textcircled{III}$
 $q_4 = q_2 a + q_3 b + q_4 a + q_4 b \rightarrow \textcircled{IV}$

$\textcircled{I} \Rightarrow q_1 = \epsilon + q_2 b + q_3 a$
 Putting values of q_2 and q_3 from \textcircled{II} and \textcircled{III}
 $q_1 = \epsilon + q_1 a b +$

$\textcircled{I} \Rightarrow q_1 = \epsilon + q_2 b + q_3 a$
 Putting values of q_2 and q_3 from \textcircled{II} and \textcircled{III}
 $q_1 = \epsilon + q_1 a b + q_1 b a$
 $q_1 = \epsilon + q_1 (ab + ba)$ $R = Q + RP$
 $R = QP^*$ *Ardens' theorem*
 $\epsilon \cdot R = R$
 $q_1 = (ab + ba)^*$
 \Rightarrow Regular Expression

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

Find the Regular Expression for the following DFA

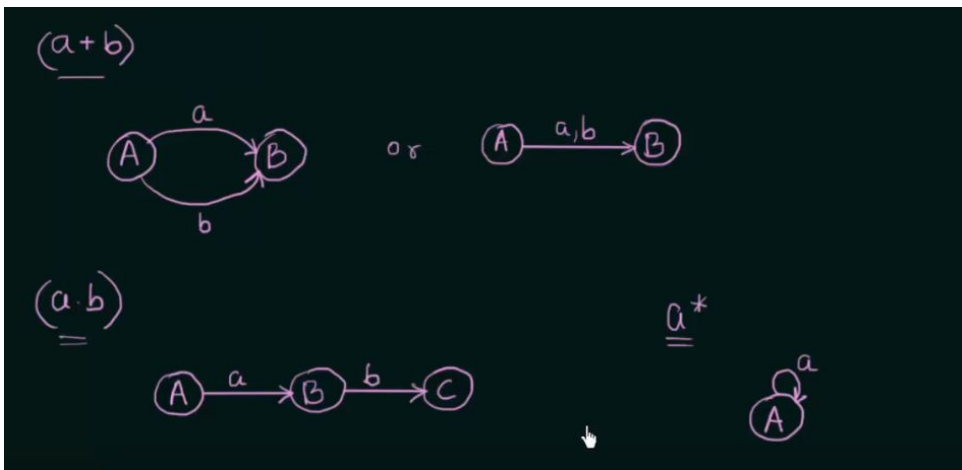
$q_1 = \epsilon + q_1 0 \rightarrow \textcircled{I}$
 $q_2 = q_1 1 + q_2 0 \rightarrow \textcircled{II}$
 $q_3 = q_2 1 + q_3 0 + q_3 1 \rightarrow \textcircled{III}$

Final state q_2

$\textcircled{I} \Rightarrow q_1 = \epsilon + q_1 0$ $R = Q + RP$
 $R = QP^*$ *Ardens' theorem*
 $\epsilon \cdot R = R$
 $q_1 = 0^*$

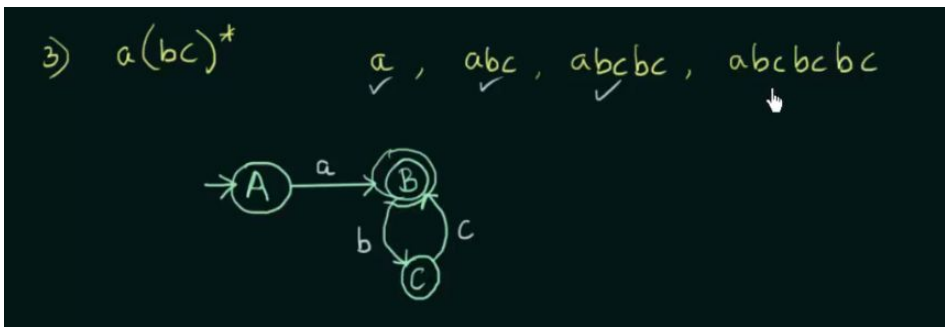
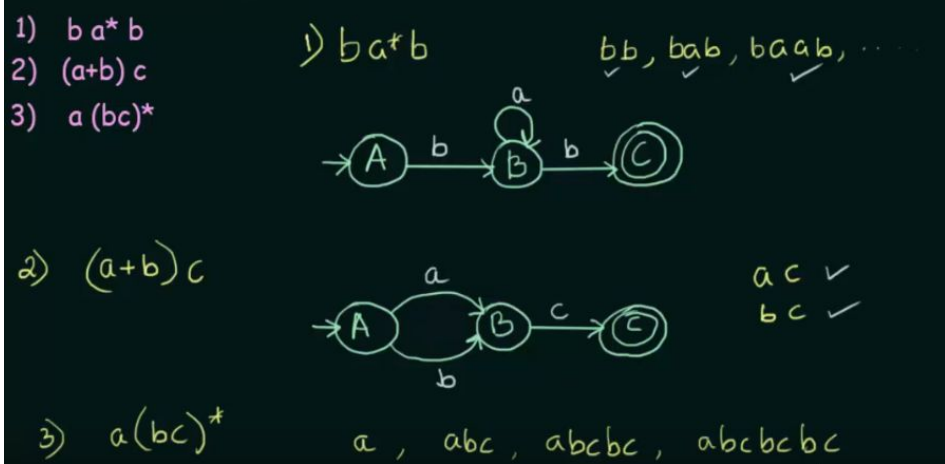
$\textcircled{II} \Rightarrow q_2 = q_1 1 + q_2 0$
 $q_2 = 0^* 1 + q_2 0$ Putting value of q_1 from \textcircled{I}
 $R = Q + RP$
 $R = QP^*$
 $q_2 = 0^* 1 (1)^*$
 $R = \text{union of both Final states}$
 $= 0^* + 0^* 1 1^*$
 $= 0^* (\epsilon + 1 1^*)$ $\epsilon + RR^* = R^*$
 $= 0^* 1^*$ \Rightarrow Regular Expression

REGULAR EXPRESSION TO FINITE AUTOMATA:



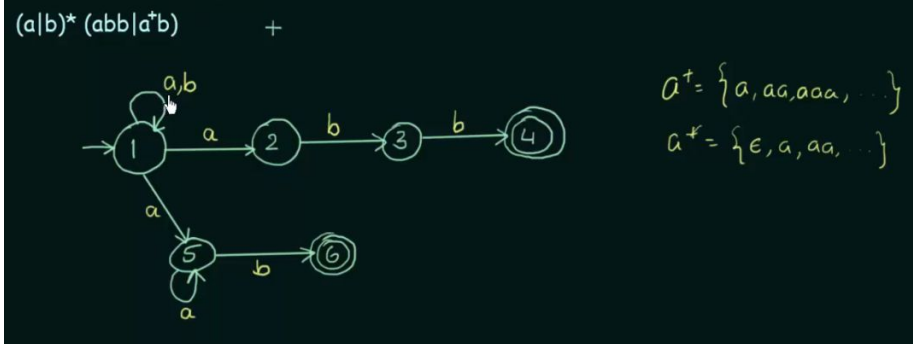
Example 1:

Convert the following Regular Expressions to their equivalent Finite Automata:



Example 2:

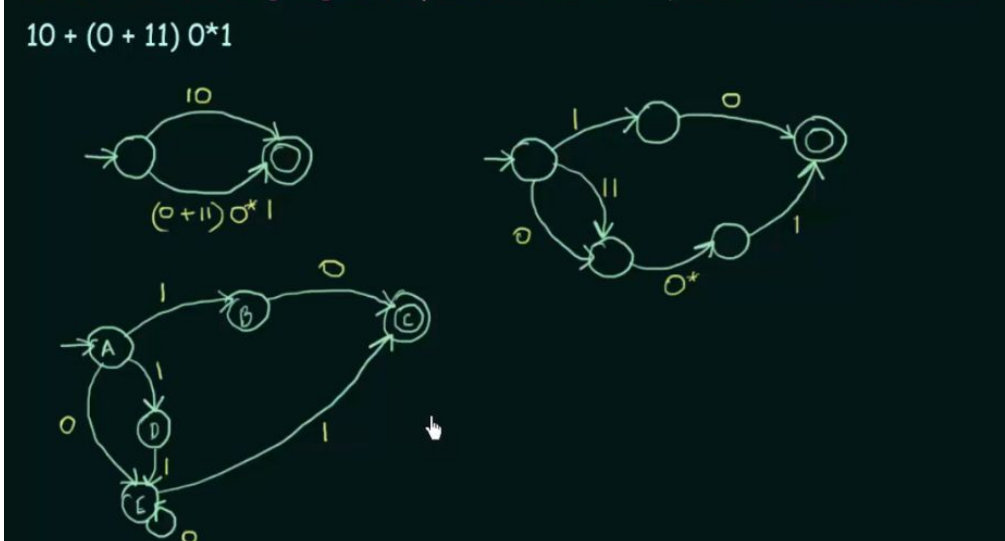
Convert the following Regular Expression to its equivalent Finite Automata:



- '|' means or.
- $(a|b)^*$ 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'.
- There are 2 finite states. Both ends with 'b'

EXAMPLE 3:

Convert the following Regular Expression to its equivalent Finite Automata:

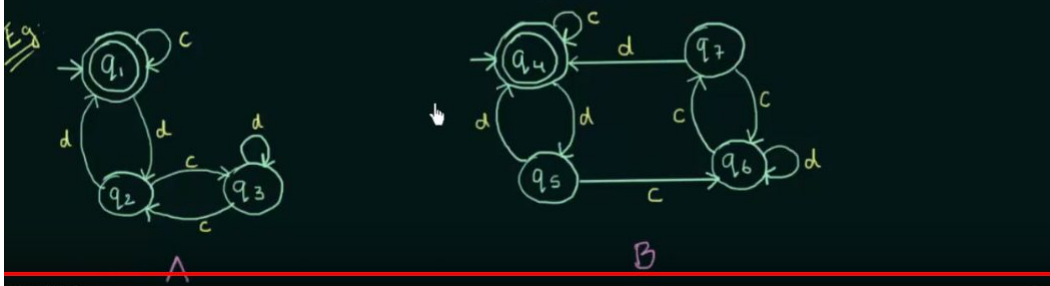


try to do i step by step

EQUIVALENT FINITE AUTOMATAS :

Example 1:

- 1) For any pair of states $\{q_i, q_j\}$ the transition for input $a \in \Sigma$ is defined by $\{q_a, q_b\}$ where $\delta\{q_i, a\} = q_a$ and $\delta\{q_j, a\} = q_b$.
The two automata are not equivalent if for a pair $\{q_a, q_b\}$ one is INTERMEDIATE State and the other is FINAL State.
- 2) If Initial State is Final State of one automaton, then in second automaton also Initial State must be Final State for them to be equivalent.



States	<u>c</u>	<u>d</u>
(q_1, q_4)	(q_1, q_4) FS FS	(q_2, q_5) IS IS
(q_2, q_5)	(q_3, q_6) IS IS	(q_1, q_4) FS FS
(q_3, q_6)	(q_2, q_7) IS IS	(q_3, q_6) IS IS
(q_2, q_7)	(q_5, q_6) IS IS	(q_1, q_4) FS FS

A and B are equivalent

Example 2:

Find out whether the following automata are equivalent or not



States	<u>c</u>	<u>d</u>
$\{q_1, q_4\}$	$\{q_1, q_4\}$ FS FS	$\{q_2, q_5\}$ IS IS
$\{q_2, q_5\}$	$\{q_3, q_7\}$ IS IS	$\{q_1, q_6\}$ FS IS

PUMPING LEMMA FOR REGULAR LANGUAGES

If A is a Regular Language, then A has a Pumping Length ' P ' such that any string ' S ' where $|S| \geq P$ may be divided into 3 parts $S = xyz$ such that the following conditions must be true:

- (1) $xy^iz \in A$ for every $i \geq 0$
- (2) $|y| > 0$
- (3) $|xy| \leq P$

Using Pumping Lemma prove that the language $A = \{a^n b^n \mid n \geq 0\}$ is Not Regular

Proof:
Assume that A is Regular
Pumping length = P
 $S = a^P b^P \Rightarrow S = aaaaaa bbbbbb$
 $P = 7$

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 $\geq P$

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

Using Pumping Lemma prove that the language $A = \{yy \mid y \in \{0,1\}^*\}$ is Not Regular

Proof: 0101
Assume that A is Regular
then it must have a pumping length = P
 $S = 0^P 10^P$
 $xy^iz \Rightarrow xy^2z$
 00000000000100000001
 $\notin A$
 $P = 7$
 0000000100000001
 $|y| > 0$
 $|xy| \leq P = 7$
 A is not Regular

EXAMPLE OF REGULAR EXPRESSIONS !!

$\Sigma = \{a, b\}$
 $L = \{aa, ab, ba, bb, aaaa, \dots\}$
 $L = \{2, 3, 4, 5, \dots\}$
 $(a+b)(a+b)(a+b)^*$
 at most 2:
 $0, 1, 2$
 $\{\epsilon, a, b, aa, ab, ba, bb\}$
 $\epsilon + a + b + aa + ab + ba + bb$
 $(a+b)\epsilon + (a+b)\epsilon$

divisible by '3'
 $L = \{0, 3, 6, 9, 12, \dots\}$
 $(a+b)(a+b)(a+b)^*$
 $\cong 2 \pmod 3$
 $(a+b)^{3n+2} / n \geq 0$
 $(a+b)(a+b)(a+b)(a+b)^*$

$\Sigma = \{a, b\}$
 even length strings.
 $L = \{\epsilon, aa, ab, ba, bb, \dots\}$
 $(a+b)(a+b)^*$
 $(a+b)^{2n} = (a+b)^{2n} / n \geq 0$

odd length \rightarrow
 $(a+b)^{2n+1} / n \geq 0$
 $(a+b)^{2n}(a+b)$
 $(a+b)^2(a+b)$
 $((a+b)(a+b))^* a + b$

starts with a:
 $a(a+b)^*$
 ends with a:
 $(a+b)^*a$
 containing a:
 $(a+b)^*a(a+b)^*$
 starting and ending with different symbols:
 $a(a+b)^*b$
 $b(a+b)^*a$

$\Sigma = \{a, b\}$
 a's exactly 2:
 b^*ab^*a
 a's atleast 2:
 $b^*ab^*a(a+b)^*$
 a's at most 2:
 $b^*(\epsilon+a)b^*(\epsilon+a)b^*$

a's are even:
 $(b^*ab^*ab^*)^* + b^*$
 $\{b, bb, bbb, \dots\}$
 $\cong (b^*ab^*a)^* \cdot b^*$
 starting and ending with same symbol:
 $L = \{\epsilon, a, b, aa, bb, \dots\}$
 $a(a+b)^*a + b(a+b)^*b + a + b + \epsilon$

no 2 a's should come together.
 $L = \{\epsilon, b, bb, bbb, a, ab, aba, abab, ababa, ba, bab, baba, babab, \dots\}$
 $(a+b)(a+b)^*$
 $(b+ab)^* + (b+ab)^*a$
 $(\epsilon, b, bb, bbb, \dots, ab, abab, \dots)$
 $(b+ab)^*(\epsilon+a)$
 $a(b+ba)^* + (b+ba)^*$
 $(a+\epsilon)(b+ba)^*$

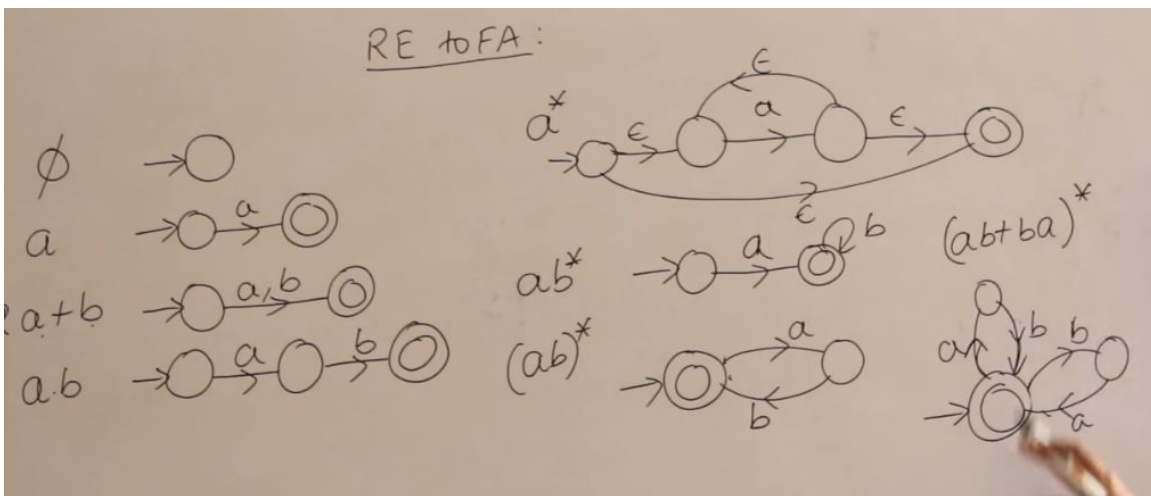
no 2 a's, no 2 b's.
 $L = \{\epsilon, a, b, ab, ba, aba, bab, \dots\}$
 $(\epsilon+b)(ab)^*(\epsilon+a)$
 $(\epsilon+a)(ba)^*(\epsilon+b)$

	Starts with	ends with
$\{a, aba, ababa, \dots\}$	a	a $(ab)^*a + a(ba)^*$
$\{ab, abab, ababab, \dots\}$	a	b $(ab)^*b + a(ba)^*b$
$\{ba, bab, babab, \dots\}$	b	a $(ba)^*a + b(ab)^*a$
$\{b, bab, babab, \dots\}$	b	b $(ba)^*b + b(ab)^*$

$(ab)^*a + (ab)^* + b(ab)^*a + b(ab)^*$
 $(ab)^*(a+\epsilon) + b(ab)^*(a+\epsilon) = (\epsilon+b)(ab)^*(\epsilon+a)$

Identities of RE
 $\rightarrow \phi + R = R + \phi = R$
 $\rightarrow \phi \cdot R = R \cdot \phi = \phi$
 $\rightarrow \epsilon R = R \epsilon = R$
 $\rightarrow \epsilon^* = \epsilon$
 $\rightarrow \phi^* = \epsilon$
 $\rightarrow \epsilon + RR^* = R^*R + \epsilon = R^*$
 $\rightarrow (a+b)^* = (a^*+b^*)^*$
 $= (a^*b^*)^*$
 $= (a^*+b)^*$
 $= (a+b)^* = a^*(ba^*)^* = b^*(ab^*)^*$

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 rule has the form $a \rightarrow \beta$ where a and β are strings on V U T and at least one symbol of a belongs to V.

Example: $G = (\{S, A, B\}, \{a, b\}, S, \{S \rightarrow AB, A \rightarrow a, B \rightarrow b\})$

$$V = \{S, A, B\}$$

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

$$S = S$$

$$P = S \rightarrow AB, A \rightarrow a, B \rightarrow b$$

Eg. $S \rightarrow AB$
 $\rightarrow aB$
 $\rightarrow \underline{ab}$

Regular Grammar:

Regular Grammar can be divided into two types:

Right Linear Grammar

A grammar is said to be Right Linear if all productions are of the form

$$A \rightarrow xB$$

$$A \rightarrow x$$

where $A, B \in V$ and $x \in T$

Eg. $S \rightarrow abs | b$ - Right linear

$$S \rightarrow sbb | b$$
 - Left linear

Left Linear Grammar

A grammar is said to be Left Linear if all productions are of the form

$$A \rightarrow Bx$$

$$A \rightarrow x$$

where $A, B \in V$ and $x \in T$

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 $G1 = (\{S, A\}, \{a, b\}, S, \{S \rightarrow aAb, aA \rightarrow aaAb, A \rightarrow \epsilon\})$

$$\begin{aligned} S &\rightarrow aAb \quad [\text{by } S \rightarrow aAb] \\ &\rightarrow aaAbb \quad [\text{by } aA \rightarrow aaAb] \\ &\rightarrow a aaAbbb \quad [\text{by } aA \rightarrow aaAb] \\ &\rightarrow a aabbb \quad [\text{by } A \rightarrow \epsilon] \end{aligned}$$

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

$$\begin{aligned} S &\rightarrow AB \\ &\rightarrow ab \end{aligned}$$

$$L(G2) = \{ab\}$$

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

$$\begin{array}{llll} S \rightarrow AB & S \rightarrow AB & S \rightarrow AB & S \rightarrow AB \\ \rightarrow ab & \rightarrow aAbB & \rightarrow aAb & \rightarrow abB \\ & \rightarrow aabb & \rightarrow aab & \rightarrow abb \end{array}$$

$$\begin{aligned} L(G3) &= \{ab, a^2b^2, a^2b, ab^2, \dots\} \\ &= \{a^m b^n \mid m \geq 0 \text{ and } n \geq 0\} \end{aligned}$$

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 $S \rightarrow 0B | 1A, A \rightarrow 0 | 0S | 1AA | \wedge, B \rightarrow 1 | 1S | 0BB$ generates the string 00110101

$$\begin{aligned} S &\rightarrow 0B \quad (S \rightarrow 0B) \\ &\rightarrow 00BB \quad (B \rightarrow 0BB) \\ &\rightarrow 001B \quad (B \rightarrow 1) \\ &\rightarrow 0011S \quad (B \rightarrow 1S) \\ &\rightarrow 00110B \quad (S \rightarrow 0B) \\ &\rightarrow 001101S \quad (B \rightarrow 1S) \\ &\rightarrow 0011010B \quad (S \rightarrow 0B) \\ &\rightarrow 00110101 \quad (B \rightarrow 1) \end{aligned}$$

Example: Verify whether the Grammar $S \rightarrow aAb, A \rightarrow aAb | \wedge$ generates the string aabbb

$$\begin{aligned} S &\rightarrow aAb \\ &\rightarrow a aAbb \quad (A \rightarrow aAb) \\ &\rightarrow a aabb \quad (A \rightarrow \wedge) \\ &\rightarrow a a aAbbb \quad (A \rightarrow aAb) \\ &\rightarrow a a aabbb \quad (A \rightarrow \wedge) \end{aligned}$$

\wedge MEANS EMPTY STRING