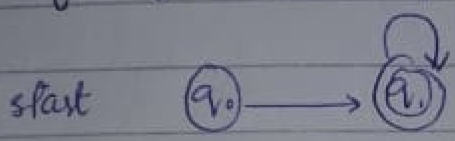
Tuesday
21/01/19

→ Deterministic: For any particular input, it gives particular output

→ Recognizable: Reached to final state.

Model for infinite possible numbers



→ Valid Machine: Accepts valid while rejects invalid.

→ lexical Analyzer: left to right scan and recognize a valid word/alphabet.
Tokenization is a technique.

Thursday ②

24/Jan/19

→ Alphabets :-

.- $\boxed{\text{Finite set}}$ of symbols or letters.

eg:

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

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

Ram → Cache

32 - bit instruction

4 bits → opcode

28 bits → address

CU → reads opcode and recognize instruction

eg: Alphabets of octal

$\{0 - 7, 10 - 17, 20, 27 - 30 \ldots\}$

eg: $\Sigma \{aB, Cd\}$ → 2 alphabets.

AlGOL → 113 letters.

Includes letters, digits and variety of operators such as GOTO and If.

→ Strings:
- Concatenation of finite symbols from alphabet.

→ Empty String: String with 0 character.
Zero occurence of alphabets.
Transition without 0 input is expressed as Empty string.
eg: Screen saver → Time is input but it happens when you do nothing.

Denoted by "λ" or 'ʌ'.

→ Words:
- They are strings (set of alphabets).
- It belongs to some language.

eg:
* String with max length 2
$\Sigma = \{a, b, ab, ba\}$.

When we impose some conditions.

"{All Words are strings but not all strings are Words}".

eg:

$L = \{x^n, n = 1, 2, 3\}$

$Y = \{\underbrace{x, xx, xxx}_{\text{words \& strings}}, \underbrace{xxxx, xxxxx}_{\text{strings}}\}$

$L = \{x^{odd}\}$

$Y = \{\underbrace{x \quad xxx \quad xxxxx}_{\text{words \& strings}}\}$

→ Valid / InValid Alphabets

Invalid → multiple symbols alphabet

$\Sigma = \{B, ab, Bab\}$ → 3 words

Eg: $\Sigma = \{B, Ba, bab, d\}$

string → BababB.

- If ambiguity is there while tokenizing a string than it is invalid.

- If an alphabet is a prefix of another alphabet in a same set it means that alphabet is invalid.

source program + input → interpreter → output.

(5)

source program → compiler → target program

$\Sigma_1 = \{B, aB, bab, d\}$ → valid

$\Sigma_2 = \{B, Ba, bab, d\}$ → invalid b/c of ambiguity.

→ length of Strings: =

- Denoted by $|S|$.
- Number of letters in strings.

eg:

$\Sigma = \{B, aB, bab, d\}$

$S = BaBbabd$

Tokenizing = (B), (aB), (bab), (d).

$|S| = 4$.

Monday.
28/Jan/19

⑥

→ Reverse of a string :=

Rewriting of tokens of a string in reverse order

eg: $L = \{10, 01\}$
$S = 100110$
$S^r = 100110$

$L = \{1, 0\}$
$S = 100110$
$S^r = 011001$

$\Sigma = \{a, b, c\}$
$S = \{abc\}$
$S^r = cba$

$\Sigma = \{B, aB, bab, d\}$
$S = BaB\,bab\,Bd$
$Rev(S) = dB\,bab\,aB\,Ba$

→ Operation on Words.

* Repitition of words upto n times

$$w^n = \underbrace{ww\cdots w}_{n}.$$

eg: $(abba)^2 = abbaabba$.

$$w^0 = \lambda$$
$$(abba)^0 = \lambda.$$

* The set of all possible string from alphabet Σ including empty string.
  * operation.

$$\Sigma = \{a, b\}$$
$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab \cdots\}$$

$$(ab)^* = \{\lambda, ab, abab, ababab, \cdots\}.$$

$$(a+b)^* = \{\lambda, a, b, ab, aba, \cdots\}.$$

$$a+b = b+a$$
$$a \cdot b \neq b \cdot a$$

$\bigcirc \xrightarrow{a} \bigcirc \xrightarrow{b} \bigcirc$ a followed by b

$\bigcirc \xrightarrow{b} \bigcirc \xrightarrow{a} \bigcirc$ b followed by a.

* + Operation.
The set of all possible strings from alphabet Σ except Λ.

$\Sigma = \{a, b\}$
$\Sigma^+ = \Sigma^* - \{\Lambda\}$
$\Sigma^+ = \{a, b, ab, aa, \dots\}$

→ Operation on languages.

A language is any subset of Σ^*
Language of all possible words

$\Sigma = \{a, b\}$
$\Sigma^* = \{\Lambda, a, b, aa, ab, \dots\}$

(Identifier → user defined names)

Identifiers → infinite language.
Keywords → finite language.

NOTE :

language which contains empty string

→ $\phi = \{\ \} \neq \{\Lambda\}$

The language contains no words.

→ Set size $|\{\ \}| = |\phi| = 0$

→ Set size $|\{\Lambda\}| = 1$

→ String length $|\Lambda| = 0$.

$L = \{a^n b^n : n \geq 0\}$.

(order)

Λ
ab
aabb
aaabbb

words
infinite Language.

abb $\notin$ L (condition satisfy nhi hori).

→ The Usual Set Operations

* $\{a, ab, aaaa\} \cup \{bb, ab\} =$
$\{a, ab, bb, aaaa\}$.

* $\{a, ab, aaaa\} \cap \{bb, ab\} =$
$\{ab\}$.

* $\{a, ab, aaaa\} - \{bb, ab\} =$
$\{a, aaaa\}$

$\bar{L} = \Sigma^* - L$

* Complement universal - language

$\overline{\{a, ba\}} = \{\lambda, b, aa, ab, bb, aaa, \ldots\}$

a, ba, aba, baa, aaba, baaa, baba

The complement operation is a subtraction operation of given language from a universal language.

→ Reverse of a Language :-

$L^R = \{W^R : W \in L\}$

$\{ab, aab, baba\}^R = \{ba, baa, abab\}$

eg: $L = \{a^n b^n : n \geq 0\}$
$L^R = \{b^n a^n : n \geq 0\}$

→ Concatenation
(Cartesian Product) { strcat (str1, str2) }

$$L_1 L_2 = \{xy : x \in L_1 , y \in L_2\}$$

$\{a, ab, ba\}\{b, aa\}$
$= \{ab, aaa, abb, abaa, bab, baaa\}$

If language is infinite then resultant will also be infinite.

→ Power operation on Language:-

$$L^n = \underbrace{LL \ldots L}_{n}$$

$\{a, b\}^3 = \{a, b\}\{a, b\}\{a, b\} =$
$\{aaa, aab, aba, abb, baa, bab, bba, bbb\}$

$L^0 = \{\lambda\}$

→

* $L^2 = \{a^n b^n \; a^m b^m : n, m \geq 0\}$

$aabbaaabbb \in L^2$.

* $L^3 = \{a^n b^n \; a^m b^m \; a^p b^p : n, m, p \geq 0\}$

→ Star Closure on languages (Kleene*) :-

$L^* = \overset{\lambda}{L^0} \cup L^1 \cup L^2$.

$\{a, bb\}^* = \lambda$
a, bb
$aa, abb, bba, bbbb$
$aaa,$

$L^1 = \{a, ab\}$
$L^0 = \lambda$
$L^2 = \{aab, aba, aa \mid abab\}$
$L^3 = \{ababab, aaa, aaba\}$

Tuesday
29/Jan/19

→ Defining Language:

1. Descriptive definition of a language:
   - A theoritical expression of language

2. Regular expressions:
   - Defining language through mathematical notation.

3. Finite Automata:
   - Models, abstract machines.

1 Descriptive definition:

- The language L of strings of odd length defined over $\Sigma = \{a\}$ can be written as :

$$L = \{a, aaa, aaaaa, \ldots\}$$

- The language L of strings that does not start with 'a' defined over $\Sigma = \{a, b, c\}$ can be written as

$$L = \{b, c, ba, \ldots\}$$

- The language L of strings of length 2 defined over $\Sigma = \{0, 1, 2\}$ can be written as:
  $L = \{01, 02, 10, \dots\}$ $(0+1+2)\{2\}$

- The language L of strings ending in '0' defined over $\Sigma = \{0, 1\}$ can be written as:
  $L = \{0, 00, 10, \dots\}$

- Language EQUAL of strings with no. of a's equals to no. of b's defined over $\Sigma = \{a, b\}$
  $L = \{\Lambda, ab, abab, abba, \dots\}$

- Language EVEN-EVEN of strings with even no. of a's and even no. of b's defined over $\Sigma = \{a, b\}$
  $L = \{\Lambda, aabb, abab, \dots\}$

- The language PRIME of strings defined over $\Sigma = \{a\}$, as $\{a^p : p \text{ is prime}\}$ can be written as
  $L = \{a, aa, aaa, \dots\}$

www.regexr.com

- Palindrome

The language consisting of Λ and the strings S defined over Σ such that Rev(S) = S.

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

$L = \{\Lambda, a, b, aa, bb, aaa, aba, \ldots\}$.

2. Regular Expressions:

It's a mathematical notation that can express Regular languages.

Language
/ \
Regular (Arithematic Progression) (difference same) Non-regular

Regular Expression is a pattern to express the language.

$*$ → zero or more occurence of alphabets

$+$ → one or more occurrence

$\cdot$ → concatenation.

? → 0, 1

[abc] → [a or b or c]

$(a+b+c) \rightarrow a \cup b \cup c$

$a\{2\} \rightarrow \{aa\}$

$a? \ \{\Lambda, a\}$

eg: $(a+b.c)^*$
describes the language
$\{a, bc\}^* = \{\lambda, a, bc, aa, abc, bca, \ldots\}$

$RE = (a+b)^*$
$L = \{\lambda, b, a, aa, bb, aba, \ldots\}$

$RE = (ab)^*$
$L = \{\lambda, ab, abab, ababab, \ldots\}$

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

NOTE:
- 'All finite languages are regular'.
- 'There are some finite languages where there is no A.P.'

Q Can we use Regular expression for finite languages with no A.P

31/Jan/19

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

Primitive Regular Expression :-

Language	Regular Expression.
ϕ	ϕ
$\{\Lambda\}$	Λ
$\{a\}$	a
$\{a,b\}^*$	$(a+b)^*$

- Concatenation of two or more regular expression is a regular expression.

$$r_1 \cdot r_2 = r$$

eg:

$$r_1 = a^*, \quad r_2 = (a+b)$$
$$r_1 \cdot r_2 = a^* \cdot (a+b)$$

- Union of two or more regular expression

$$r_1 + r_2 = r$$

eg:

$$r_1 = a^*, \quad r_2 = b^*$$
$$r_1 + r_2 = r$$
$$a^* + b^*$$

- Closure of regular expression is a regular expression

$$r = (r)^*$$

eg:

$$r = (a+b)$$

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

→ $(a+b)^* = \{\lambda, a, b, ab, ba, aab, \ldots\}$

→ $a^* + b^* = \{\lambda, a, b, aa, bb, aaa, bbb, \ldots\}$

→ $(a+b)^* = (b+a)^*$

→ $(ab)^* \neq (ba)^*$

→ $(ab)^* = \{\lambda, ab, abab, ababab, \ldots\}$

→ $(ba)^* = \{\lambda, ba, baba, \ldots\}$

- The language L contains all strings with atleast two consecutive zereos defined over $\Sigma = \{0, 1\}$.

  $L = \{00, 000, 001, 100, \ldots\}$

  $R.E = (0+1)^* 00 (0+1)^*$

- The language L contains all strings without two consecutive zeroes defined over $\Sigma = \{0, 1\}$.

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

$R.E = (1^{*}011^{*})^{*}(0+\Lambda) + 1^{*}(0+\Lambda)$.

* For one language there may exist more than than one regular expressions but for each regular expression there exist a one language.

- $R.E : L(r) = \{a^{2n}b^{2m}b : n, m \geq 0$.

$L = \{b, aab, bbb, \ldots\}$.

The language L of strings ending in b defined over $\Sigma = \{a, b\}$.

$R.E = (aa)^{*}(bb)^{*}b$

- $a^{2n}b^{3n}c$, $n, m \geq 0$

$R.E = (aa)^{*}(bbb)^{*}c$.

Q Is there any case when S^{+} contains Λ ?

Ans. If $S = \{\Lambda, a\}$

$S^{+} = \{\Lambda, a, aa, aaa, \ldots\}$

Yes, if set contains alphabet 'Λ'

- $(S^{+})^{*} = (S^{*})^{*}$

- $(S^{+})^{+} = S^{+}$

$\circ - (S^*)^+ = (S^+)^*$

Ex 3.1 :

① $L((a+b)^* b (a+ab)^*) < 4$.

$= \{ b, aba, abab, bba, bbab \}$

② $((0+1)(0+1)^*)^* 00 (0+1)^*$

Yes

④ $\{ a^n b^m : (n+m) \text{ is even} \}$

R.E : $(aa)^*(bb)^* + (aa)^* a (bb)^* b$

⑤ (a) $L_1 = \{ a^n b^m , n \geq 4, m \leq 3 \}$

R.E : $aaaaa^* \cdot (\lambda + b + bb + bbb)$

(b) $L_2 = \{ a^n b^m : n < 4 , m \leq 3 \}$

R.E : $(\lambda + a + aa + aaa) \cdot (\lambda + b + bb + bbb)$

$\Rightarrow$ (c) The complement of L_1

R.E : $(\lambda + a + aa + aaa + aaaa) \cdot bbbb^* + (a+b)^*$

(9) $L = \{a^n b^m : n \geq 1, m \geq 1, nm \geq 3\}$

R.E : $a \cdot a^* \cdot bbb + aaa \cdot b \cdot b^*$

(12) $L = \{vwv : v, w \in \{a,b\}^*, |v| = 2\}$.

$aa(a+b)^* aa + ab(a+b)^* ab + ba(a+b)^* ba + bb(a+b)^* bb$

4/Feb/19

→ The language of string defined over $\Sigma = \{a, b\}$ having exactly 'aa'

$$b^* a a b^*$$

→ The language L of strings with 2 a's over $\Sigma = \{a, b\}$.

$$b^* a b^* a b^*$$

→ The language L of string atleast 2 a's over $\Sigma = \{a, b\}$

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

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

→ The language L of strings with atleast one 'a' and one 'b'.

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

→ The language L of strings with even length.

$$(a+b) \{2n\}$$
$$((a+b)(a+b))^*$$

→ The language L of strings with odd length

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

→ The language L of strings startething with 'a' and ending with 'b' or starting with 'b' and ending with 'a'

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

→ The language of strings with words not ending with 'a'.

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

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

→

- [abcd] means (a|b|c|d)
- [b-g] means [bcdefg]
- [b-gM-Qxr] means [bcdefg

→ RE for Tokenization.

Identifier

letter → (a|b|c| .... |z|A|B|C| .... |Z)

digit → (0|1|2| .... |8|9)

id → letter (letter |digit)*

numbers

integer → (+|-|ε)(0|(1|2| .... |9) digit*)

decimal → integer . (digit)*

real → (integer|decimal) E (+|-) digit*

complex → '(' real ',' real ')'

if

- [a-z][a-z0-9]*
- [0-9]+
- ([0-9]+ "." [0-9]*)|([0-9]* "." [0-9]+)
- ("-" [a-z]* "\n")|(" "|"\n"|"\t") no token just white spaces.

Chapter # 2

→ Deterministic Finite Automata And Regular Languages.

Automaton : An abstract machine which performs some thing automatically.

Deterministic : specific output

Automaton : Rejects or accepts string.

DFA → combination of 5 tupels

Thursday
7/Feb/19

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

Q = finite set of states (atleast one state)
q_0 = initial state ($q_0 \in Q$)
F = set of final states ($F \subseteq Q$)
Σ = finite set of input alphabet
δ = transition function.

$$Q = \{q_0, q_1, \dots, q_n\}$$

Finite : B/c total no. of states should be finite in DFA.

No additional memory.

Regular Expression : Generator

Finite Automaton : Acceptor

Input → Output

String → Finite Automaton → Accept (OR) Reject

Accept:

* When input consumed completely & we are at final state

Reject:

* Input is consumed & we are not at final state (OR)
* We are not at final state & input is also not consumed.

- For every state, there is a transition for every symbol in the alphabet.
- Final state could be intermidiary state
- To accept a string : Whole input string is scanned & the last state (q_{Final}) is accepted

$$q_{Final} \in F.$$

- To reject a string:

$$q_{last} \in (Q - F)$$

(OR) $q_{last} \notin F$

- Transition Function: $(q \to q_0)$

$$\delta : Q \times \Sigma \to Q$$
$$\delta(q, x) = q'$$

- Extended Transition Function:

$$\delta^* : Q \times \Sigma^* \to Q$$
$$\delta^* : (q, w) = q'$$

- Inductive Definition:

Basis: $\delta^*(q, \Lambda) = q$

Induction: $\delta^*(q, w\delta) = \delta(\delta^*(q, w), \delta)$

eg: $\delta^*(q_0, bbb)$

$= \delta(\delta^*(q_0, bb), b)$

$= \delta(\delta(\delta^*(q_0, b), b), b)$

$= \delta(\delta(\delta(\delta^*(q_0, \Lambda), b), b), b)$

$= \delta(\delta(\delta(q_0, b), b), b)$

$= \delta(\delta(q_1, b), b)$

$= \delta(q_2, b)$

$= b$

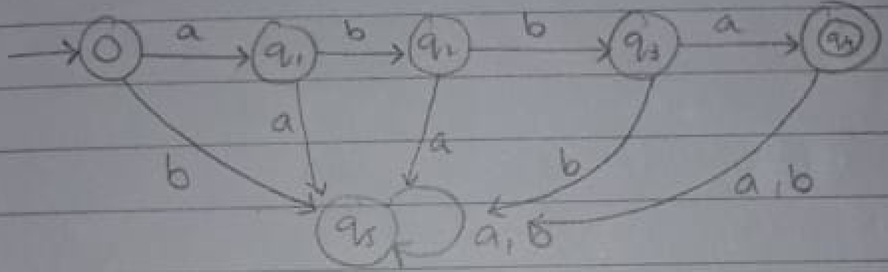
→ language of DFA:

All the strings that derive M to a final state is the langaage of DFA.

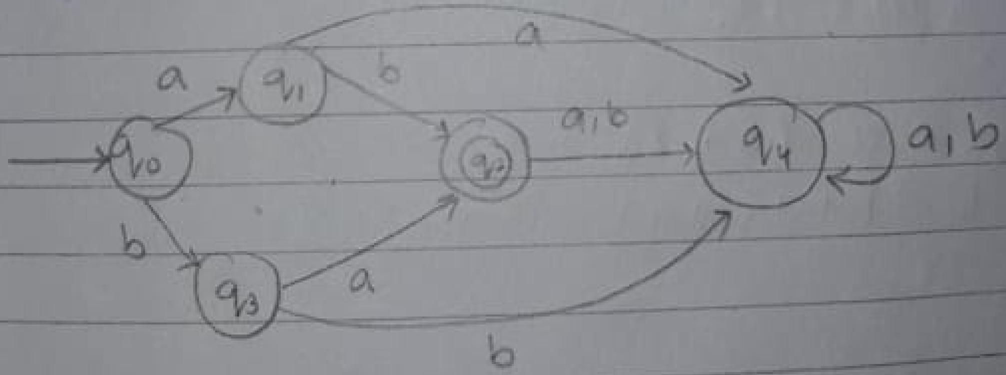
(OR)

All the strings accepted by the model is the language of the DFA.

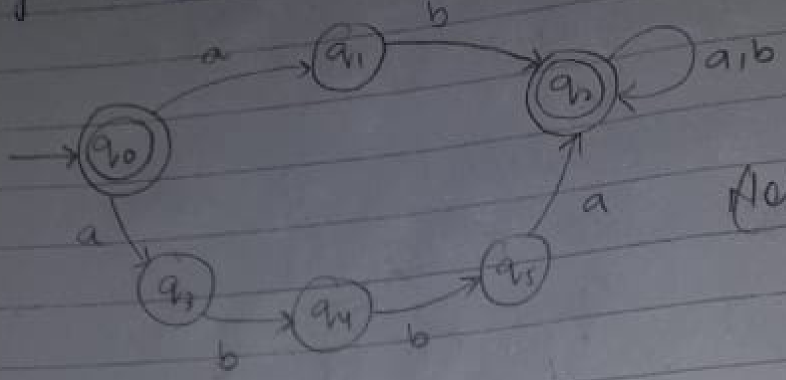
eg: L(M) = {Λ, abba}



eg: L(M) = {ab, ba}

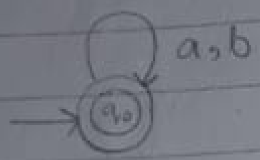


eg: L = {K, ab, abba}

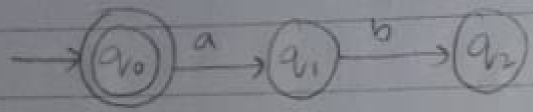


(Not a DFA)

eg: R.E : $(a+b)^*$

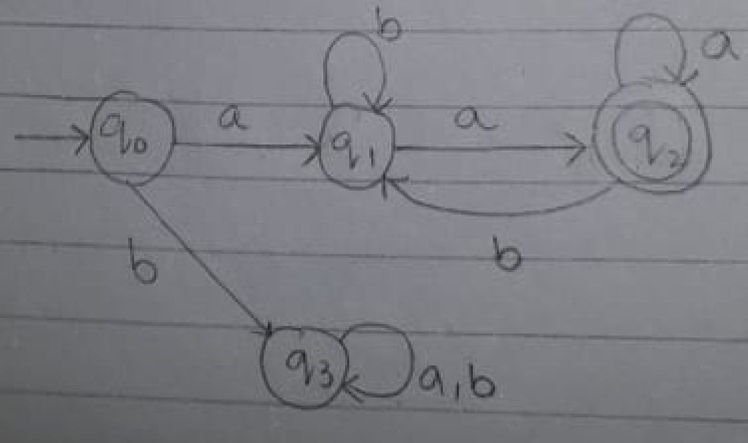


eg: Empty language



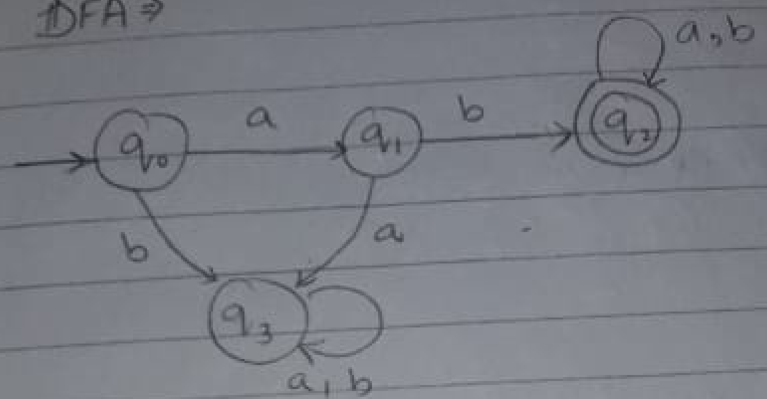
eg: $a \cdot (a+b)^* \cdot a$

L = {aa, aaa, aba, aaba, ...}



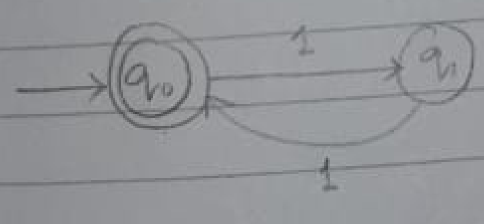
* L = all strings with prefix ab
  $L = \{ab, aba, abb, abab, \dots\}$
  $R.E = ab(a+b)^*$

DFA ⇒



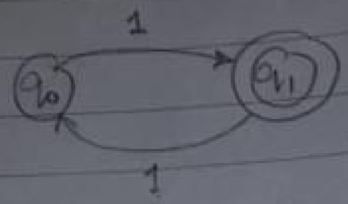
<u>NOTE</u>: For every regular expression, there exist one or more DFA's. If there exist a DFA for a language then the language is Regular.

* $L = \{x \in$ even length of strings $x \in \Sigma^*\}$
  $\Sigma = \{1\}$
  $L = \{\wedge, 11, 1111, 111111, \dots\}$



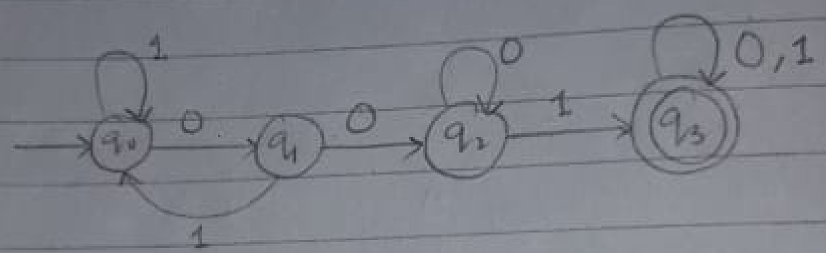
* L - {x ∈ odd length strings}

L = {1, 111, 11111, }



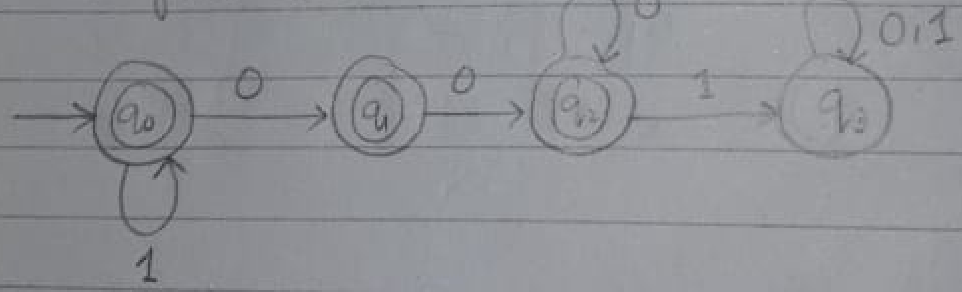
* L = {all binary strings containing substring 001}

R.E : $(0+1)^*\ 001\ (0+1)^*$

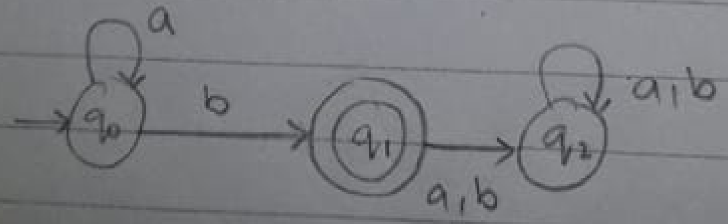


* L = {all binary strings without substring 001}

Hint : Take complement and change initial & final states.

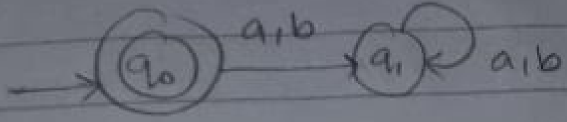


* L = $\{a^n b : n \geq 0\}$ R.E : a^*b



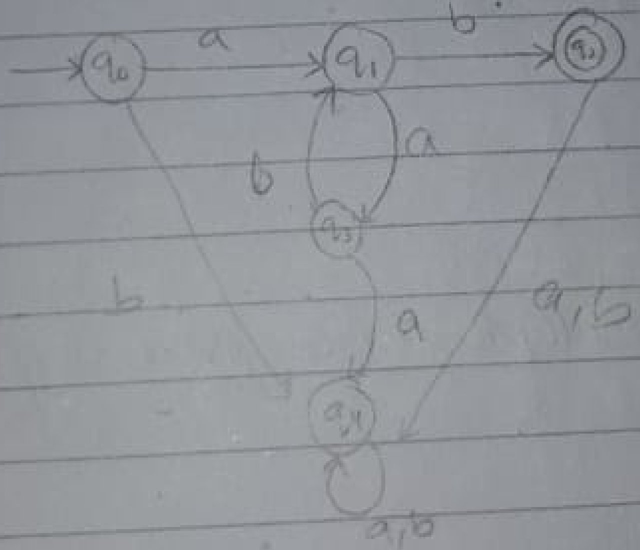
* $L = \{\ \}$ ⇒ Empty language

* $L(M) = \{\lambda\}$

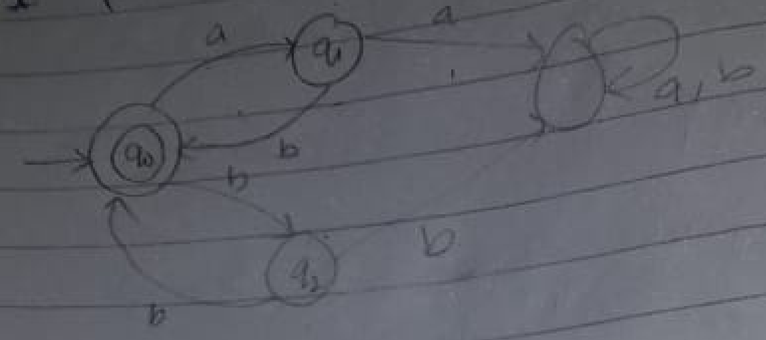


* $L = \{a^n b^n : n \geq 0\}$

* $a.(ab)^*.b$



* $(ab + bb)^*$

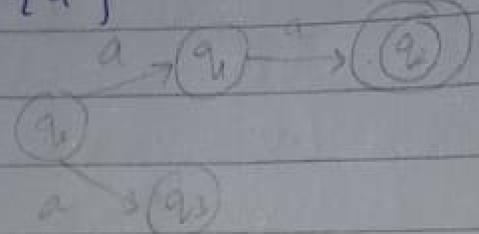


→ Non - Deterministic Finite Acceptor (NFA).

Simpler to design

difficult to program.

$\Sigma = \{a\}$



* May have multiple choice for a single input.

NFA accepts a string:
If whole string is consumed and automaton is at final state.

NFA rejects a string:

NFA definition:

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

→ Extended Transition Function = δ^*

1) Base clause : $\delta^*(q_0, \wedge) = \{q\}$

2) Inductive clause : $\delta^*(q, ya)$
$= \bigcup_{p \in \delta^*(q,y)} \delta(p, a)$

Example:

$\delta^*(0, ab) = ?$

Sol

$\Rightarrow \bigcup_{p \in \delta^*(0,a)} \delta(p, b)$

$\Rightarrow \delta^*(0,a) = \bigcup_{p \in \delta^*(0,\wedge)} \delta(p, a)$

$\Rightarrow \delta^*(0,\wedge) = \{0\}$

$\Rightarrow \delta(\{0\}, a) = \{0, 1, 3\}$

$\Rightarrow \delta(\{0,1,3\}, b) = \delta(0, b) \cup \delta(1, b) \cup \delta(3, b)$

$= \{2\} \cup \{3\} \cup \{1\}$

$= \{1, 2, 3\}$

$\Rightarrow \delta^*(0, ab) = \{1, 2, 3\}$

Every DFA is also NFA

 Every NFA is also NFA-Λ

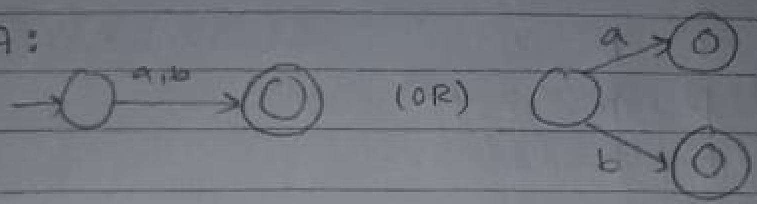
14/Feb/19

→ NFA with Null Transition:-

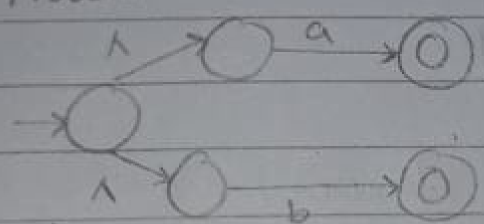
$L = \{a, b\}$, $\Sigma = \{a, b\}$, R.E = $a+b$

DFA:

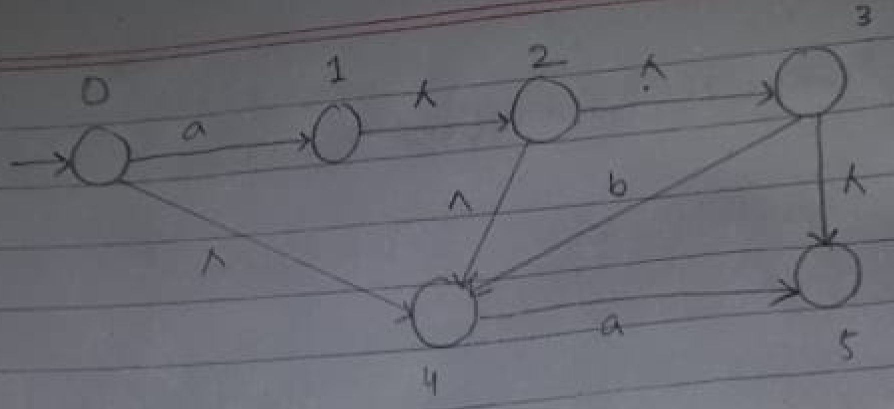
NFA:



NFA NULL:



If a language can be accepted by DFA, than it ~~can~~ is also be accepted by NFA & NFA-Λ.



states	a	b	λ
0	1	φ	4
1	φ	φ	2
2	φ	φ	3, 4
3	φ	4	5
4	5	φ	φ
5	φ	φ	φ.

$\delta(0, \lambda) = \{4\}$

→ Definition of λ-closure:

"λ closure of set S of states of Q by λ(S)" (OR) λ(S) is defined as the number of states that can be reached by reading λ arc, including S itself.

$\lambda(0) = \{0, 4\}$

$\lambda\{1\} = \{1, 2, 3, 4, 5\}$

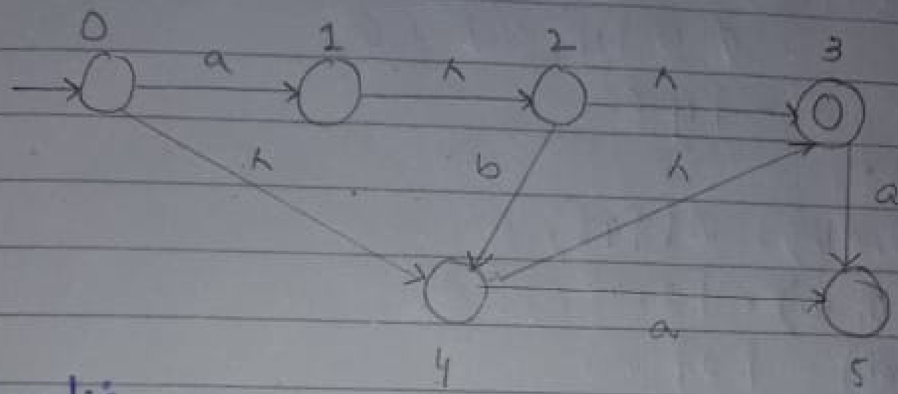
$\lambda\{2\} = \{2, 3, 4, 5\}$

$\lambda\{3\} = \{3, 5\}$.

→ Extended transition function:

Definition of δ^*:

- Base clause: $\delta^*(q, \Lambda) = \Lambda(\{q\})$.
- Inductive clause: $\delta^*(q, ya) = \Lambda\left(\bigcup_{p \in \delta^*(q,y)} \delta(p, a)\right)$



Example:

$\delta^*(0, ab) = ?$

Sol:

$\delta^*(0, ab) = \Lambda\left(\bigcup_{p \in \delta^*(0,a)} \delta(p, b)\right) \rightarrow (1)$

$\delta^*(0, a) = \Lambda\left(\bigcup_{p \in (\delta^*(0,\Lambda))} \delta(p, a)\right) \rightarrow (2)$

$\delta^*(0, \Lambda) = \Lambda(\{0\}) = \{0, 3, 4\}$.

Putting this in eq (1)

$\delta(\{0,3,4\}, a) = \delta(0, a) \cup \delta(3, a) \cup \delta(4, a)$

$= \{1\} \cup \{5\} \cup \{5\}$

$= \{1, 5\}$

Putting this in eq (2)

$\delta^*(0, a) = \Lambda(\{1, 5\})$

$= \Lambda\{1\} \cup \Lambda\{5\}$

$= \{1, 2, 3\} \cup \{5\}$

$\delta^*(0, a) = \{1, 2, 3, 5\}$

Putting this in eq 1

$\delta(\{1, 2, 3, 5\}, b) = \delta(1, b) \cup \delta(2, b) \cup \delta(3, b) \cup \delta(5, b)$

$= \phi \cup \{4\} \cup \phi \cup \phi$.

$= \{4\}$

$\delta^*(0, ab) = \Lambda(\{4\})$

$\delta^*(0, ab) = \{3, 4\} \cap \{3\}$

$\delta^*(0, ab) = \{3\}$.

$L = \{\Lambda, a, ab\}$

$R.E = \Lambda + a + ab$.

→ Conversion of NFA-Λ to NFA:

NFA-Λ.

→ null-closure of q

① State (q)	② Input (Σ)	$\Lambda(\{q\})$	$\bigcup_{p \in \Lambda(\{q\})} \delta(p,\sigma)$
0	a	{0,1}	$\delta(0,a) \cup \delta(1,a)$ $= \phi \cup \{1,2\} = \{1,2\}$
0	b	{0,1}	ϕ
1	a	{1}	{1,2}
1	b	{1}	ϕ
2	a	{2}	ϕ
2	b	{2}	{3}
3	a	{1,3}	{1,2}
3	b	{1,3}	ϕ

③ $\delta_2(q,a) = \Lambda\left(\bigcup_{p \in \Lambda(\{q\})} \delta(p,\sigma)\right)$

{1, 2}

ϕ

{1, 2}

ϕ

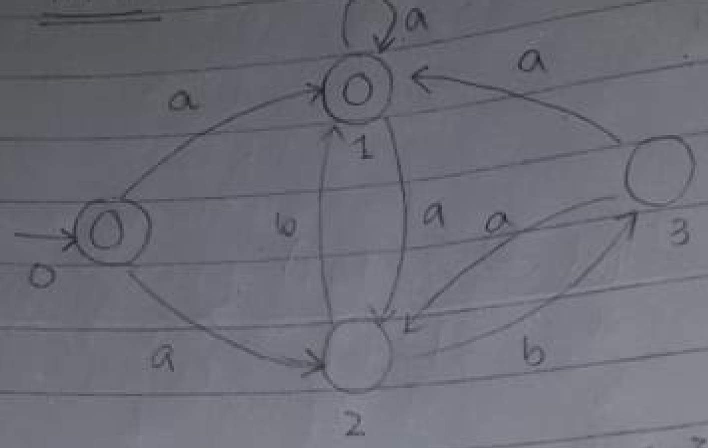
ϕ

{1, 3}

{1, 2}

ϕ

NFA:



$A_1 = \{1\}$ $\{0,1\} \cap \{1\} = \{1\}$

if (Λ $\{0\} \cap A_1 \neq \phi$) then

$A_2 = A_1 \cup \{0\} \rightarrow A_2 = \{0,1\}$

else

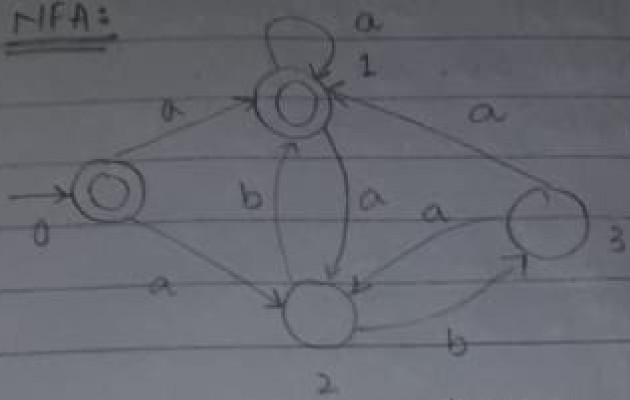
$A_2 = A_1$

$L = \{\Lambda, a, ab, aab, aaba, \ldots\}$

$R.E = a^* (ab)^* a^*$

→ Conversion from NFA to DFA :-

NFA:

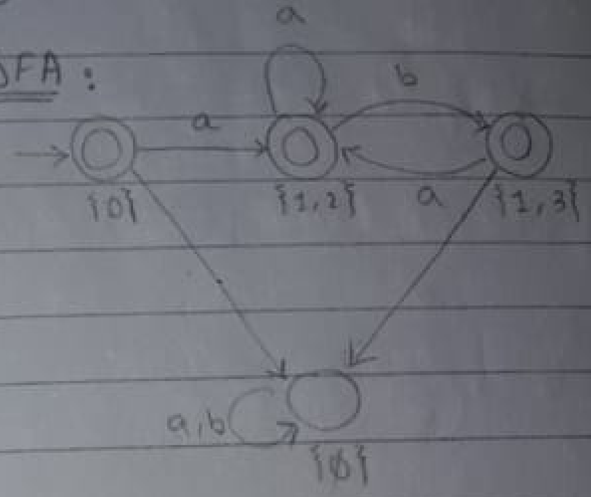


L = {Λ, a, ab, aab, ...}

R.E = $a^*(ab)^*a^*$

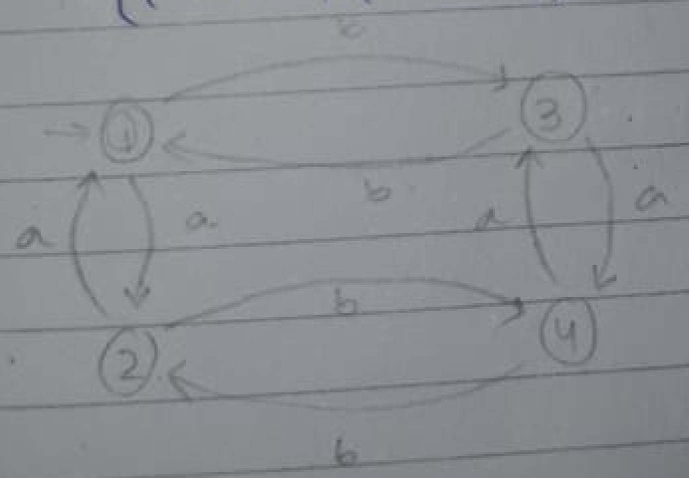
State	a	b
{0}	{1,2}	ϕ
{1,2}	{1,2}	{1,3}
{1,3}	{1,2}	ϕ
{ϕ}	ϕ	ϕ

DFA :



→ Even- Even example

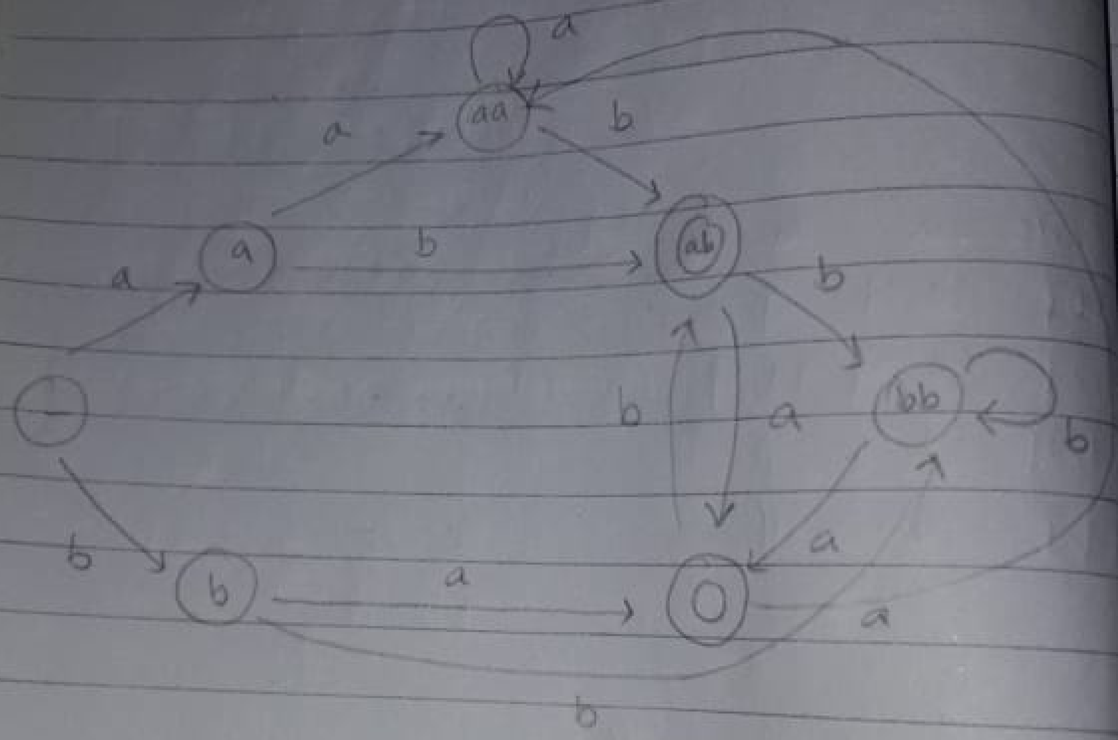
$[(aa+bb)(ab+ba)]^*$



19/Feb/19

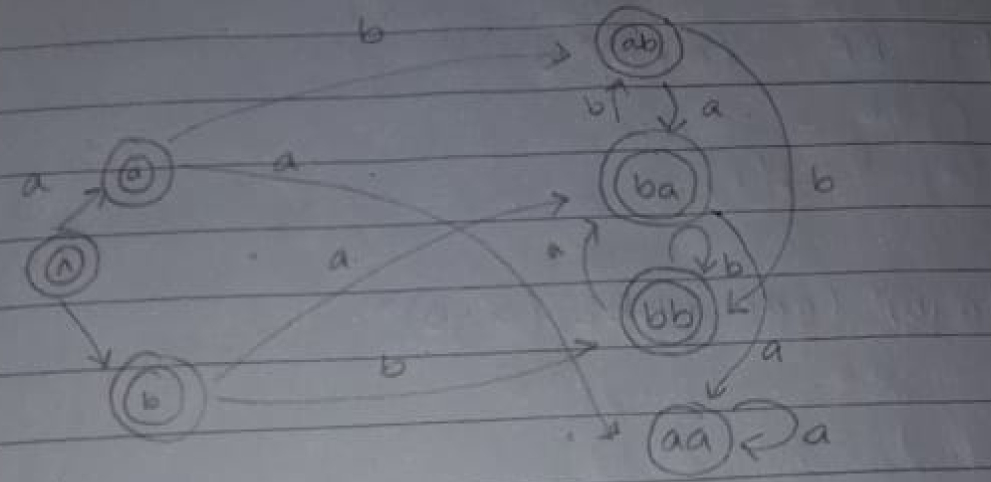
→ John Martin

→ L = {w belongs to {a,b}*: length(w) >= 2 &
w neither ends in aa nor bb}.
= (a+b)*(ab+ba).



→ $L = \{w$ belongs to $\{a, b\}^*$, w does not end in $aa\}$.

$R.E = \Lambda + a + b + (a+b)^* (ab + ba + bb)$.

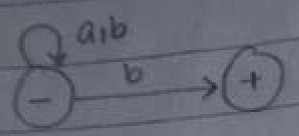


→ Transition Graph:

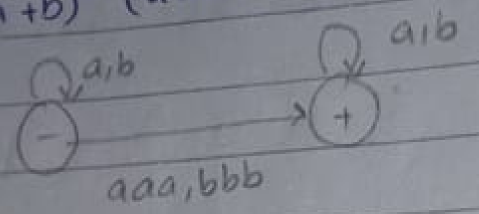
→ To read substrings.

Every FA is also TG but not every TG is FA.

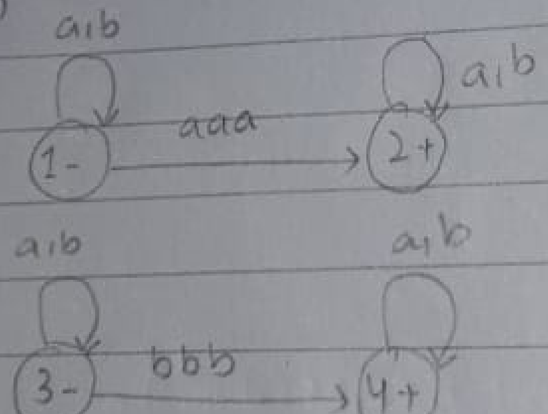
eg: RE: $(a+b)^* b$



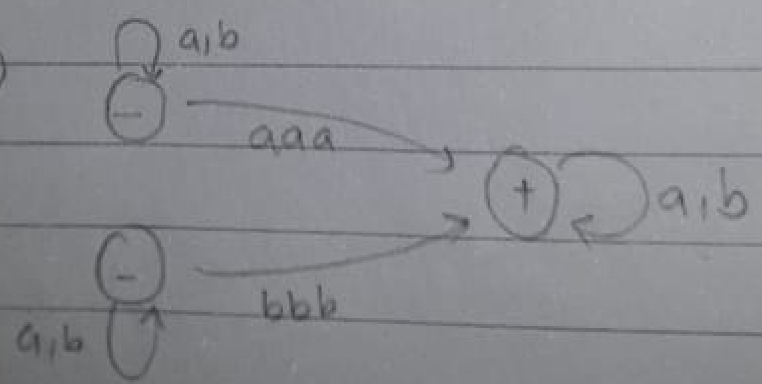
eg $(a+b)^* (aaa+bbb)(a+b)^*$



(OR)

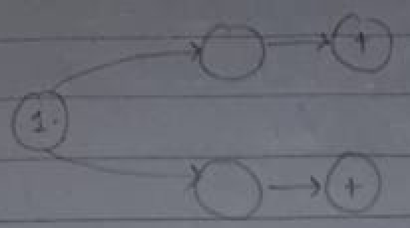


(OR)



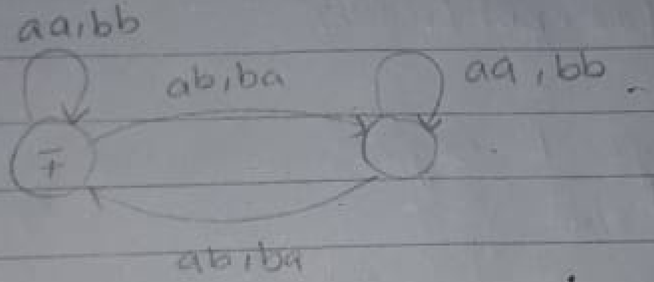
eg $a(a+b)^*b + b(a+b)^*a$

Language L beginning & ending over different alphabet



eg: Language L of Even-Even

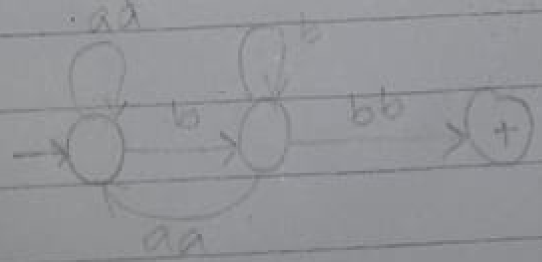
$(aa+bb + (ab+ba)(aa+bb)^*(ab+ba))^*$



eg: Language L defined over Σ{a,b} in which 'a' occurs in even combination and ends with three or more b's.

$(aa)^*b(b^*+(aa(aa)^*b)^*)bb$.

(OR) $(aa)^*b(b^*+((aa)+b)^*)bb$.

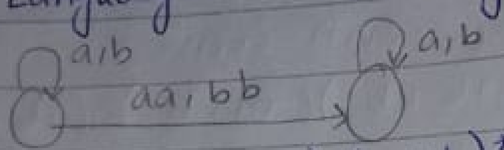


Thursday
21/Feb/19

→ Generalized Transition Graph:

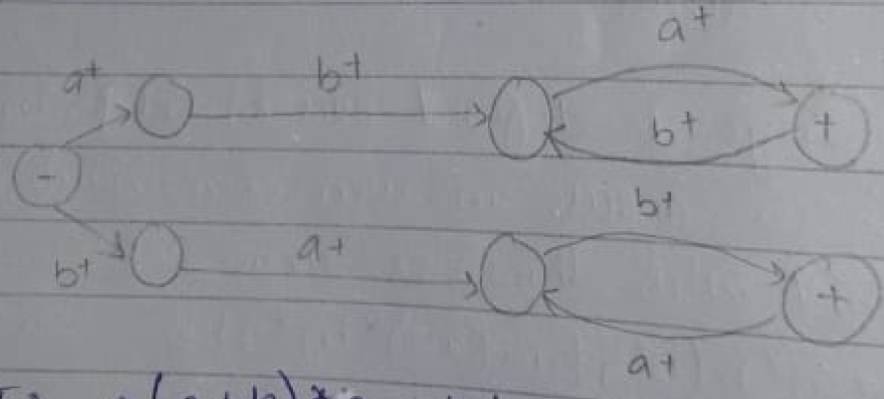
① Finite no. of sets.
② Finite set of input letters (Σ) from which input string formed.
③ Directed edges connected some pair of states labeled by R.E.

eg Language L containing aa or bb.



R.E: $(a+b)^*(aa+bb)(a+b)^*$.

eg# Language L beginning and ending with same letter.



R.E: $a(a+b)^*a + b(a+b)^*b$.

→ Kleene's Theorem :-

If a language is expressed by FA, TG, RE then it can also be expressed by other two as well.

Part 1: If accepted by FA, then it can be accepted by TG.

Part 2: If accepted by TG, then it can be accepted by RE.

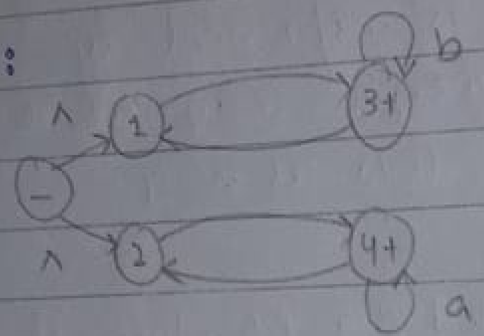
Part 3: If accepted by RE, then it can be accepted by FA.

→ Part 1: Every FA is also a TG.
(conversion not required).

→ Part 2: Given TG, extract FA.

Obtaining RE from TG.

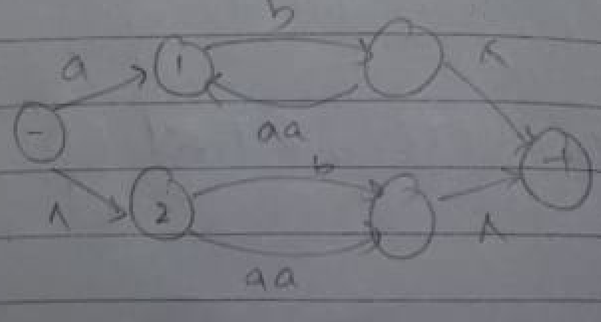
Step 1:



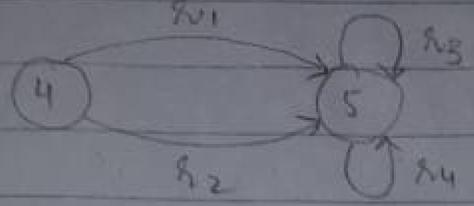
- Required if multiple initial states are there, then make new initial state connected new state by old by connecting through null.

Step 2:

More than 1 final state, then introduce a new final state by joining new state with old by null transition.



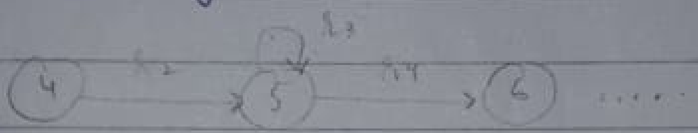
Step 3 : Reduce states



⇒

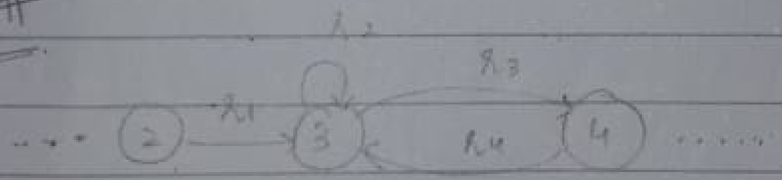
Step 4 : By Pass and state elimination.

eg#

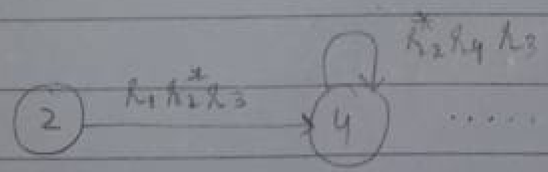


⇒

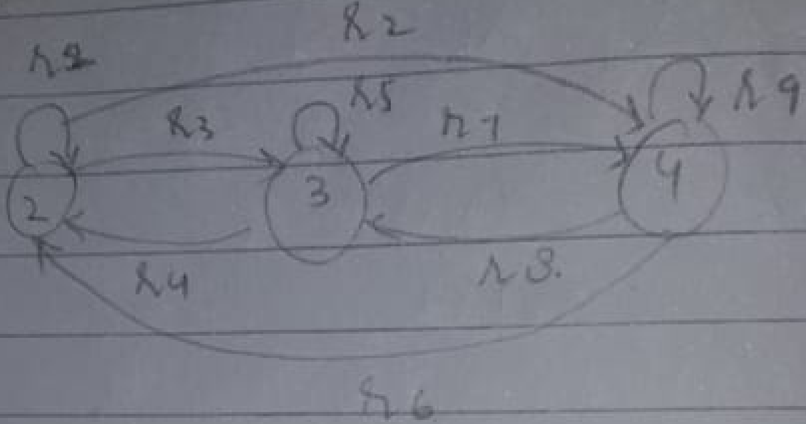
eg#



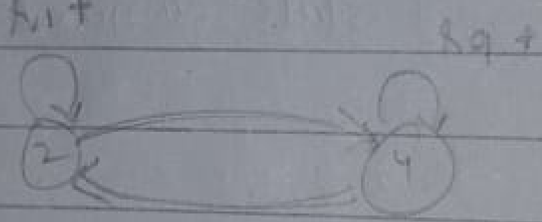
⇒



eg #



=>



$r_1 + (r_3$