- We define a language with recursive formula/method.
- Regular Expression is a valid expression when:
  1. Fullfil all language expressions
  2. Do not violate the language conditions

Operators

1. Kleene star (*) → Repeation Operator (starts with / have null ↓ (0 ...) (a, aa, ...)) => a^*
2. Kleene plus (+) → Repeationtion Operator (1 ...) => a^+
3. Concatenation (.) → Contatinate strings/symbols => (a.b) => ab
4. Option/choice operator (|) or (+) => (a+b) or (a | b)

$\Sigma = \{a, b\}$ → Language set

1. Language which produce strings that only contains a's. => {a, aa, aaa, ...}
   → R.E : a^+
2. Language which produce strings that contains a's or b's. => {a, b, aab, abb, aaab, ...}
   R.E : $(a+b)^+$
3. Language that produces combinations or a's and b's
   R.E : $a^+ \mid b^+$ or $a^+ + b^+$

(Some General Notations)

$a^+ \Rightarrow \{a, aa, aaa, \cdots\}$

$a^* \Rightarrow \{\lambda, a, aa, aaa, \cdots\}$

$(a+b)^+ \Rightarrow \{(a+b), (a+b)^2, (a+b)^3 \cdots\}$

$\Rightarrow \{(a+b)[(a+b)(a+b)][(a+b)(a+b)(a+b)]\cdots\}$

$(a+b)^* \Rightarrow \{(a+b)^0, (a+b)^1, (a+b)^2, (a+b)^3, \cdots\}$

$= \{\lambda, (a+b)^+\} \Rightarrow \{\lambda, (a+b)[(a+b)^2], [(a+b)^3 \cdots\}$

$a^+ + b^+ = \{a, aa, aaa, \cdots\} + \{b, bb, bbb, \cdots\}$

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

$(a^* + b^*)^+ = \{[\{\lambda, a, aa, \cdots\} + \{\lambda, b, bb, \cdots\}]^+\}$

$(a^* + b^*)^*$ $\qquad$ $(a+b)^*$ $\qquad$ $(a^*, b^*)^*$ $\qquad$ λ^*

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

① Start and end with same letter

$0.(0+1)^*.0 + 1.(0+1)^*.1 + 0 + 1$

Produce same "0" same "1" for single 0 or 1

② Start and end with ~~same~~ different letter

$0.(0+1)^*.1 + 1(0+1)^*.0$

③ Include atleast 2 zero

$(0+1)^*.0.(0+1)^*.0.(0+1)^*$

④ Include even number of 0

$(1^*.0.1^*.0.1^*)^* + 1^*$

⑤ Include even number of 0 and 1

$((00+11)^* + (01+10)(01+10)^*(00+11))^*$

⇒ Compilers works on regular expressions

int a = 123;

Use R.Es ⇐ How compiler knows this is valid integer?

① Construct RE for a language that doesn't contains substring ab

{λ, a, b, bb, aa, bbb, bba, baa, ...}

R.E : $a^*b^* \; b^*a^*$

② Construct RE for a language that doesn't contain substring aa

{λ, a, b, ba, bb, abb, abab, baba, bab, ...}

R.E : $(a+ab)^* \; (a+b)^* aab(a+b)^* \; (a+b)^* ((aa)^* b(aa)^*)^* \; (b+ab)^* + a + \lambda$

Operator Precedence

① * ② . ③ + ④ | or + → choice operator

→ Left to Right

R.L defines RE

③ No of a's and b's are equal

- (cannot create a regular expression because this is not a valid) ⇒ $(ab/ba)^*$

Σ ⇒ {a, b}

① Σ and λ are by default part of your language

② If R1 and R2 ∈ R.L then $(R1)^*$, $(R1).(R2)$ and R1 + R2 produces the string that are part of R.L

$a^+ = a.a^* \Rightarrow a^+$ | This is why a^+ is not included in above |

Tuesday ToA 30-1

Finite Automata

① Deterministic F.A (DFA)

② Non-Deterministic F.A (NFA)

DFA = { Σ, Q, F, q, δ } → delta (Transition ())

Σ → set of all inputs

Q → Total states

F → Final state (can be more than 1)

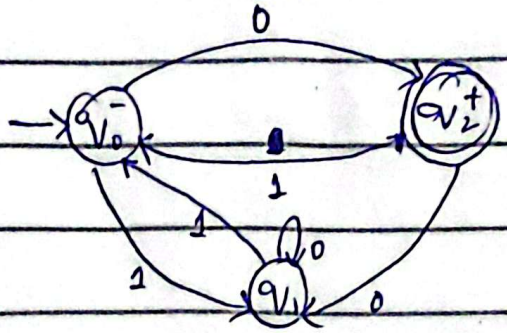
q → Initial State (only one)

$\Sigma = \{0, 1\}$, $Q = \{q_0, q_1, q_2\}$, $F = \{q_2\}$, $q_0 \rightarrow$ Initial state

State \ Σ	0	1
q_0	q_2	q_1
q_1	q_1	q_0
q_2	q_1	q_0

→ Pictorial form

- Final state is represented by '+' and initial state is shown by '-' on top of it



- If the model ends on final state then the model accepts the input.

- If the model does not ends on final state and the input ends then the model rejects the input (ends on other states)

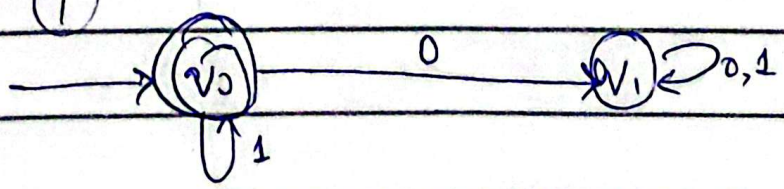
NFA

- If have no or more then one transition on any input or state.

- If the input is λ (Null) then initial state is final stage inorder to accept the null input.

$\Sigma = \{\lambda, 1, 11, 111, \ldots\}$

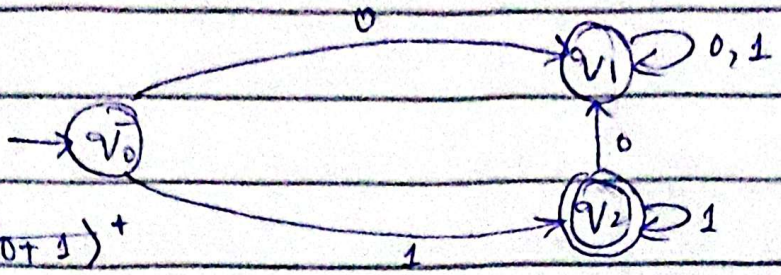
(1)



This only accepts λ and 1's

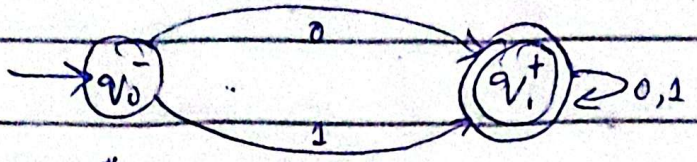
R.E : 1*

(2)

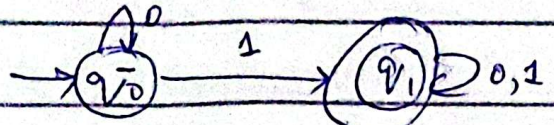


R.E : $(0+1)^+$

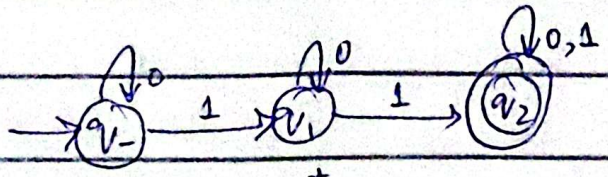
(3) R.E : $(0+1)^+$



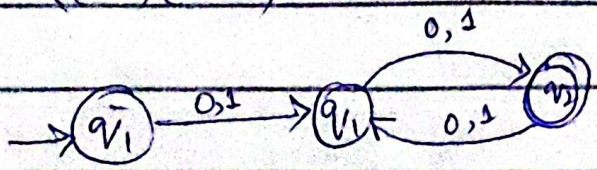
(4) R.E : $(0+1)^*.1.(0+1)^*$



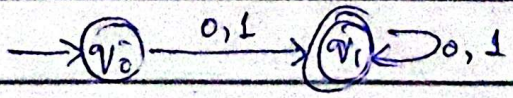
(5) R.E : $(0+1)^*.1.(0+1)^*.1$



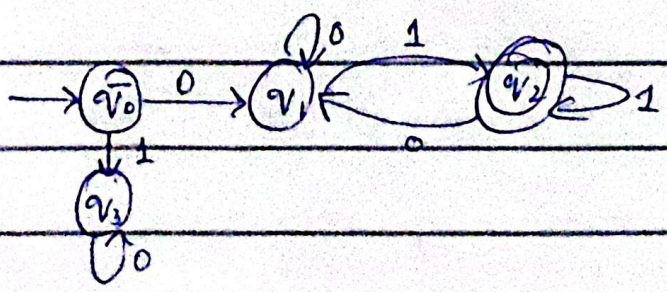
R.E : $((0+1)(0+1))^+$



R.E : $(0+1)^+$

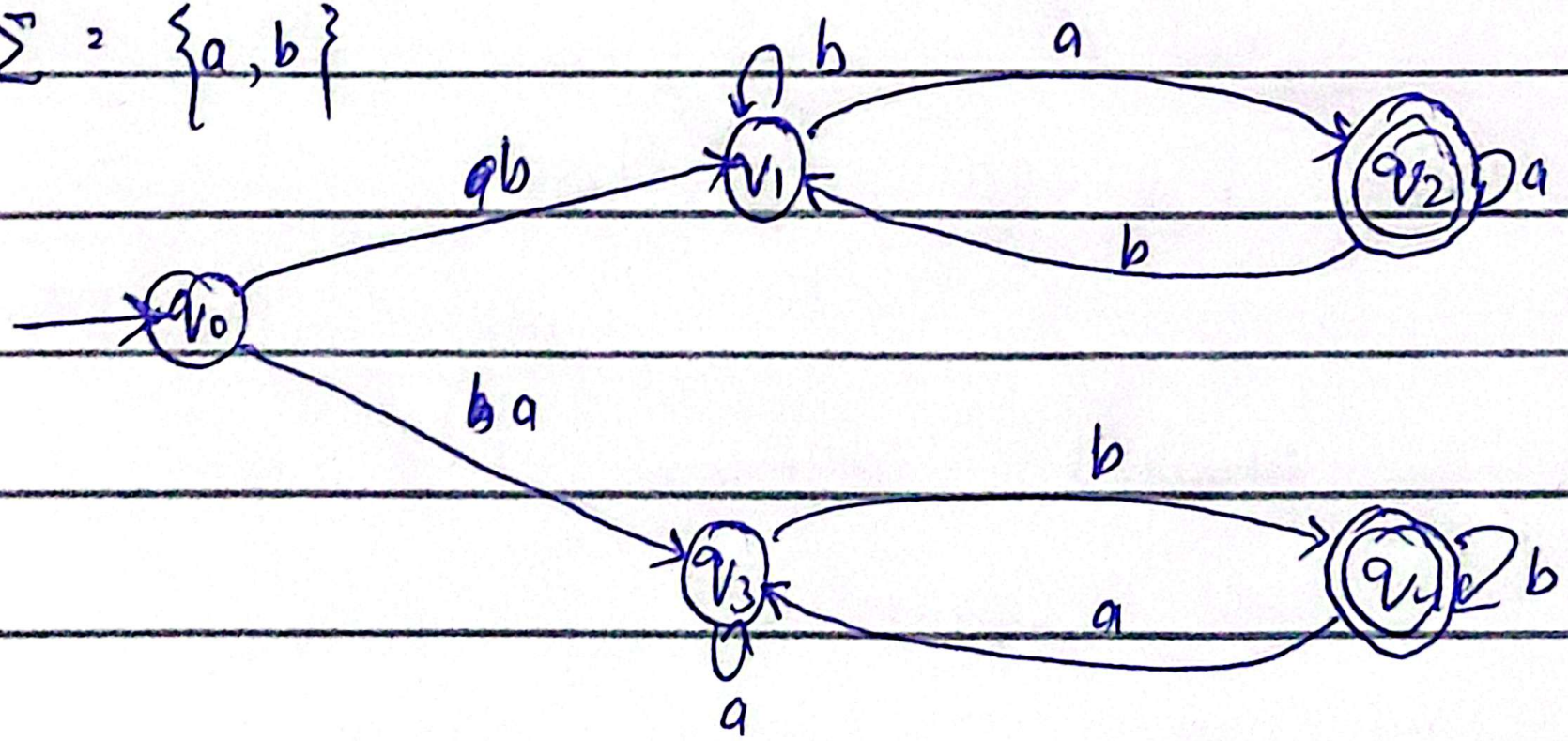


R.E : $0(0+1)^*1$

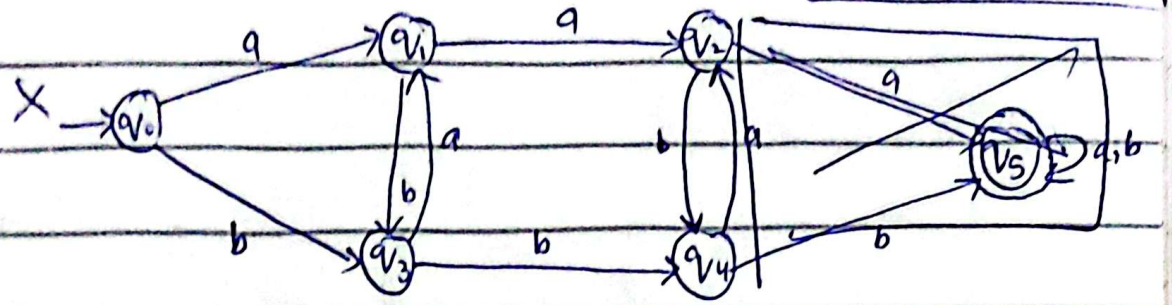


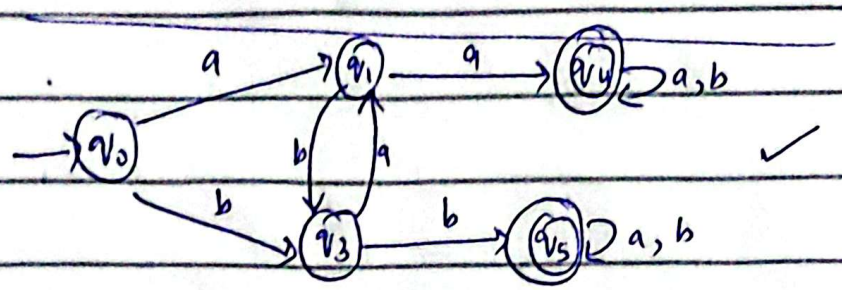
① Start and end on different symbols

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

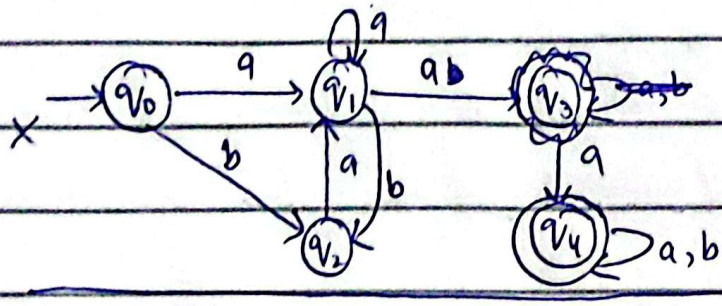


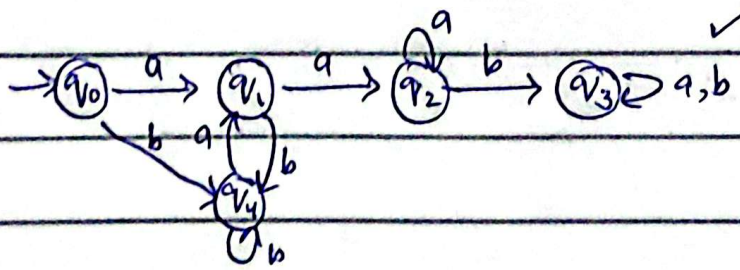
(2) Atleast 2 a's and 2 b's $(a+b)^*(aa+bb)(a+b)^*$

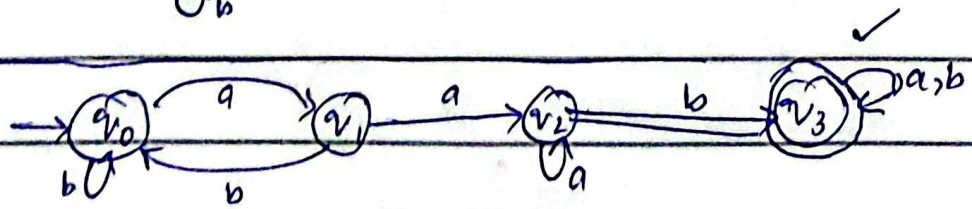




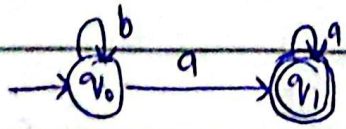
(3) $(a+b)^* aab (a+b)^*$

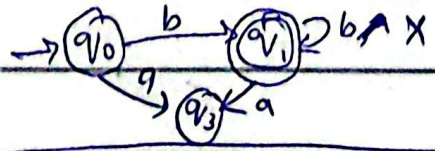




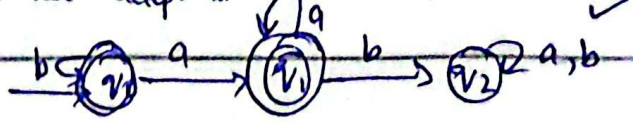


(4) ~~a+b~~ $b^* a^*$





does not accept 'ab'



① DFA ② NFA (More than one transition, and can skip a transition)

NFA ← Transition Graph

NFA Flexibility

NFA → cannot be implemented in real life.

NFA cannot map on DFA, it should be converted.

NFA = $\{\Sigma, Q, v_0, F, \delta\}$

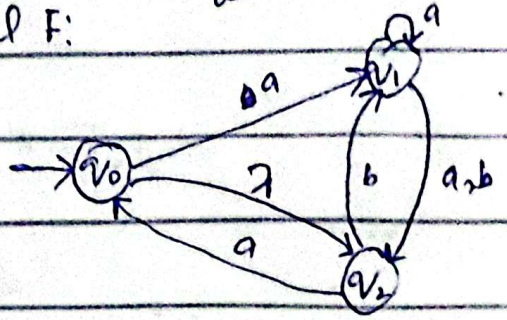
$\Sigma = \{a, b\}$, $Q = \{v_0, v_1, v_2\}$, v_0 = Initial state

$F = \{v_1\}$ → You can move to other state with λ

δ =

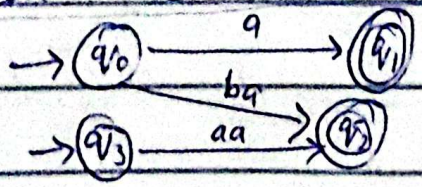
	a	b	λ
v_0	v_1	∅	v_2
v_1	$v_1 v_2$	v_2	
v_2	v_0	v_1	

Pictorial F:

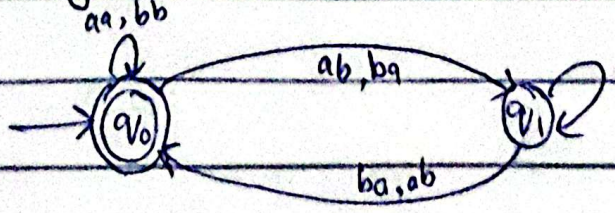


Transition Graph: we can move from one state to another using complete String rather than character. Input

- [can have more than one initial state]



① Design a machine that accepts even a's & b's



→ String can be single character, null or any length

Every DFA is by default NFA but not vice versa

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

R.E : $ab^*(ac+ba)c$

~~NFA~~ :

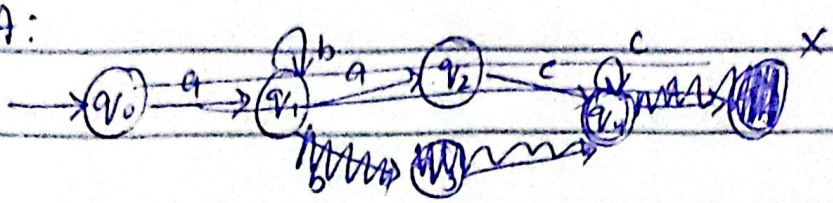
input : aacc, abac

input : ababacc, abbbac

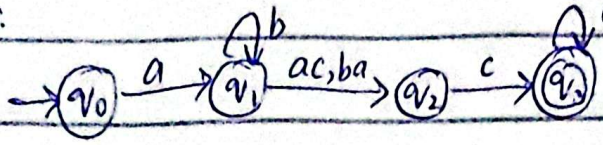
input : abbbacc, abbbbac

input : ab

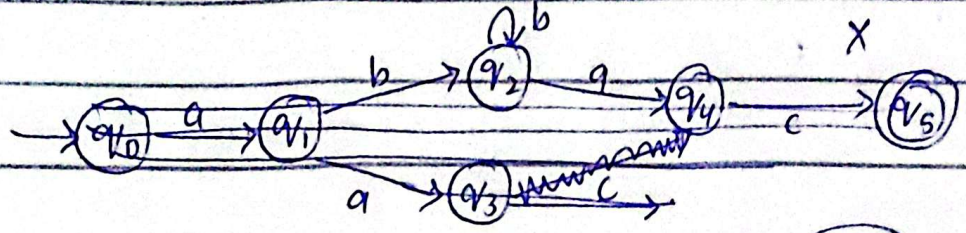
DFA:

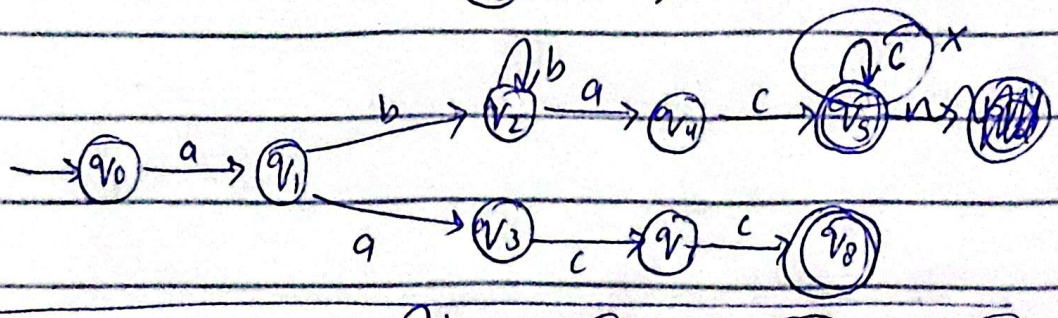


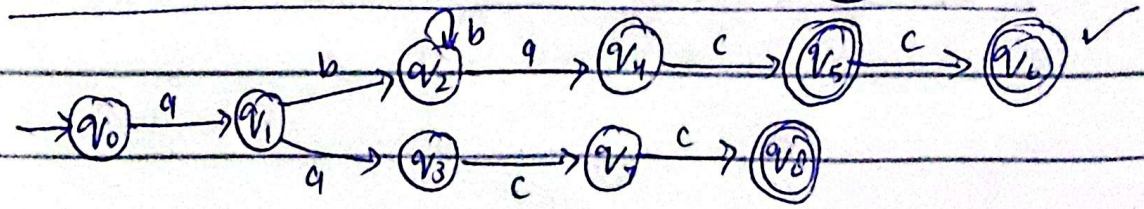
NFA:

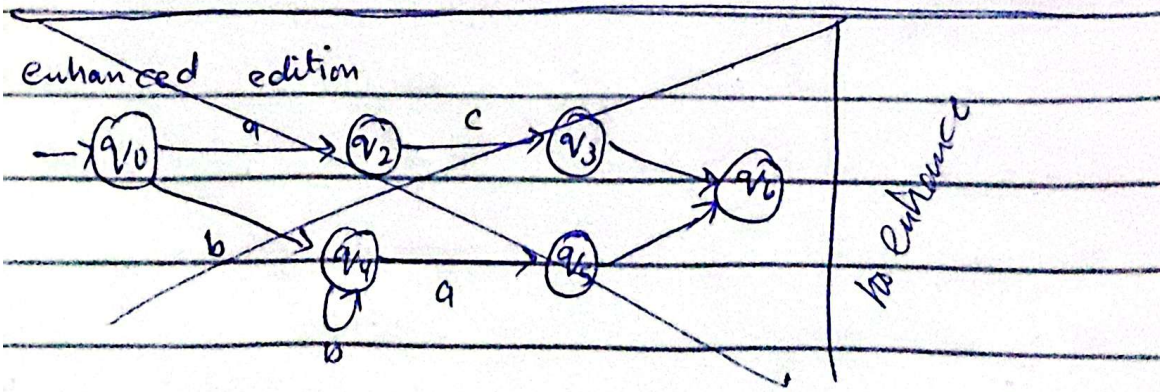


DFA:

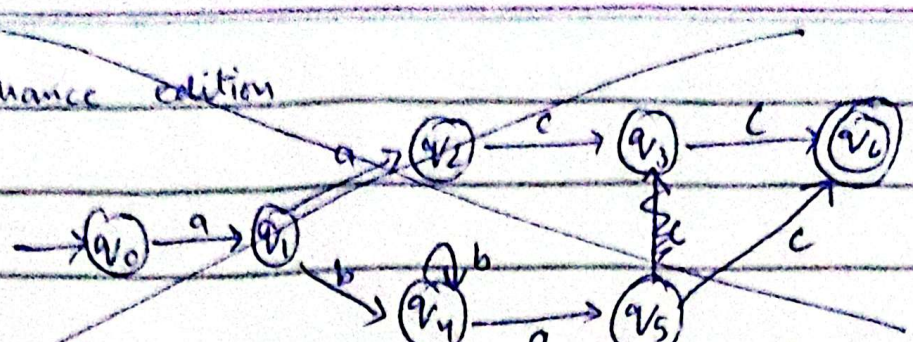




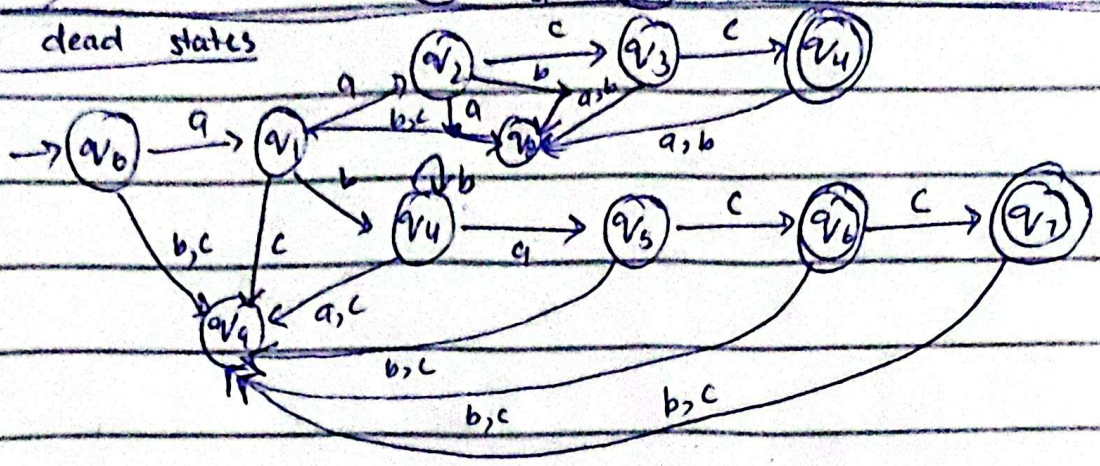




Enhance edition



with dead states



Regular Language is:

can make R.E → DFA → NFA → T.G } Kleen's Theorm — 3 parts

(i) T.G → R.E

(ii) R.E → NFA / DFA { Convert NFA to DFA }

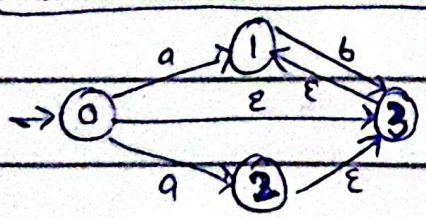
① ε, λ- Closure of every step → where can move with null transition

② Proceed with initial state closure and name it.

③ Associate S_0 with every input

④ Check the closure of associated state and also name it.

(iii) NFA → T.G



ε - closure {0} = {0, 1, 3} => S_0

ε - closure {1} = {1}

ε - closure {2} = {1, 2, 3}

ε - closure {3} = {1, 3}

Draw DFA using these ⑤ Empty state have self loops

closure ↓ Transition ↓

$(S_0, a) \Rightarrow \{1, 2\} \Rightarrow \{1, 2, 3\} \Rightarrow S_1$ + Final State

$(S_0, b) \Rightarrow \{3\} \Rightarrow \{1, 3\} \Rightarrow S_2$

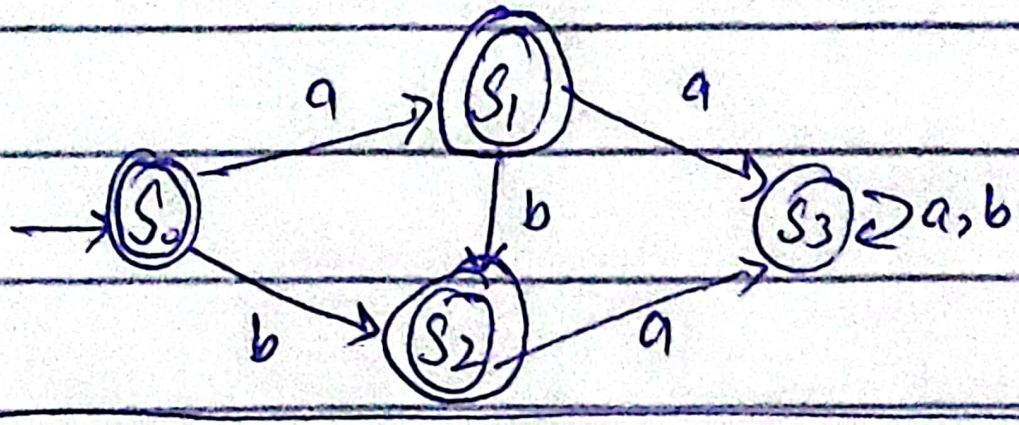
$(S_1, a) \Rightarrow \{\} \Rightarrow S_3$

$(S_1, b) \Rightarrow \{3\} \Rightarrow S_2$

$(S_2, a) \Rightarrow \{\} \Rightarrow S_3$

$(S_2, b) \Rightarrow \{3\} \Rightarrow S_2$

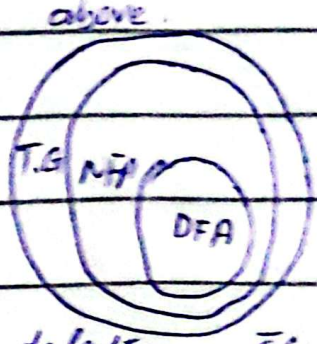
DFA :



Monday TOA 19-2

- Kleene's Theorem:

If regular expression is defined in any of (i) TG (ii) RE (iii) F.A Then it can be convertable into any of above.

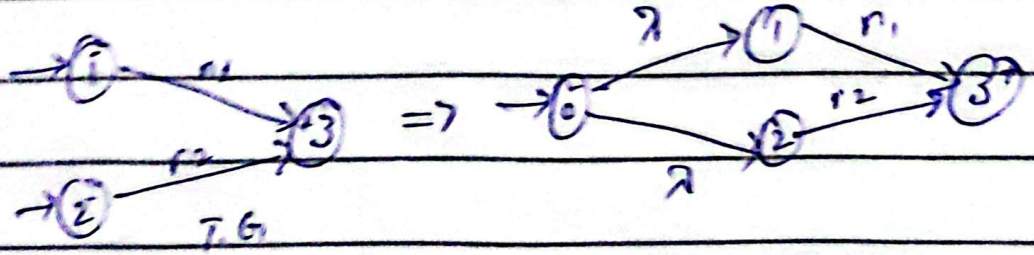


(I) F.A → T.G

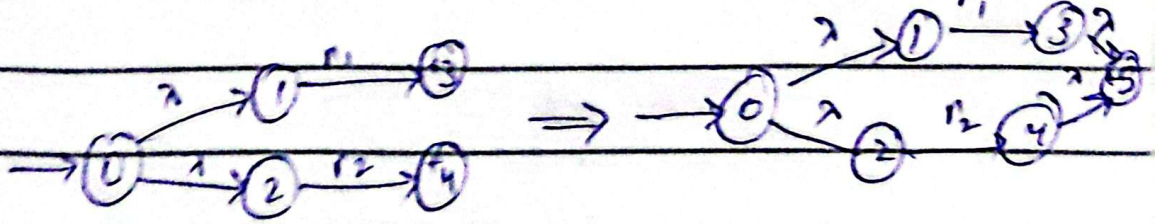
(II) T.G → R.E

(III) R.E → F.A

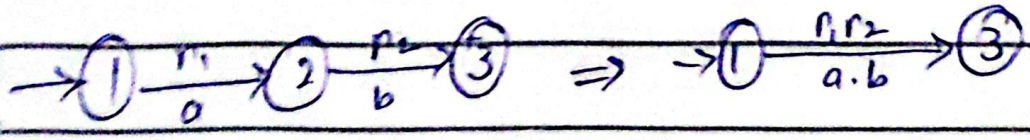
- NFA and DFA are by default TG.
- T.G have multiple initial states (can have)
- How to resolve → (Multiple initial & final state)



- 2 ~~fin~~ initial states => one initial state



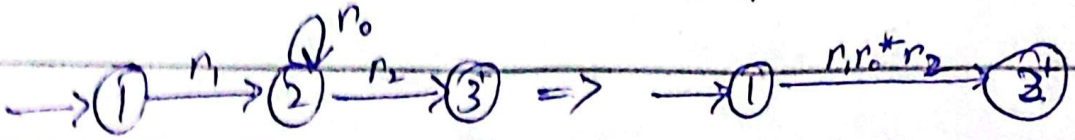
- 2 final states => one final state



- Remove multiple states => By ~~Not~~ Concatination

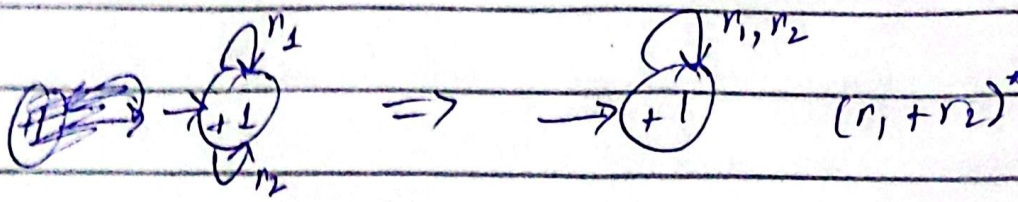
- Make sure that $r_1 r_2$ accepts the same input as r_1 and r_2.

If (2) have a self loop than

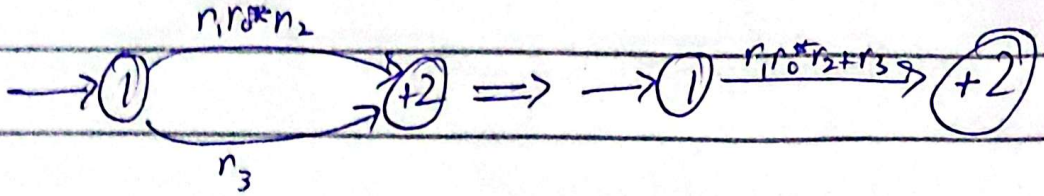


To remove self loop use (r^*) in the regular expression

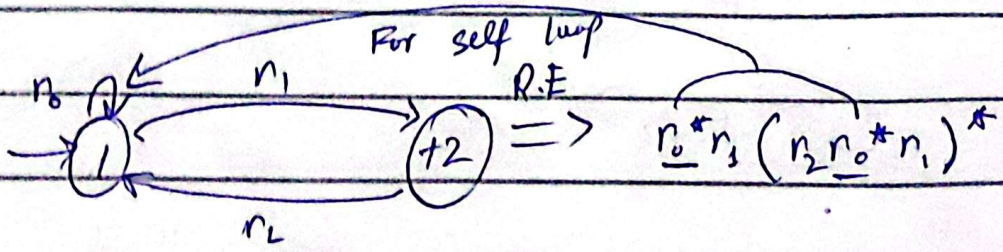
- Multiple self loops can be represented as single transition.



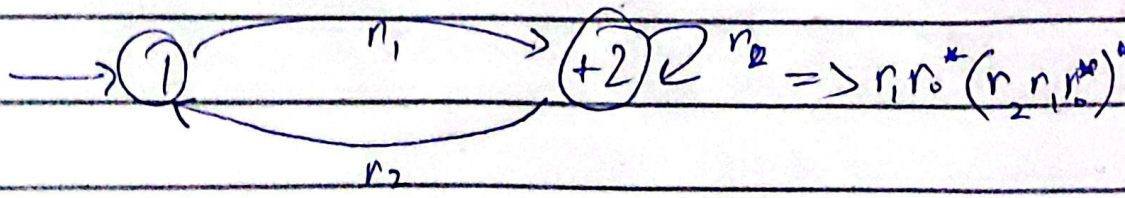
(1)



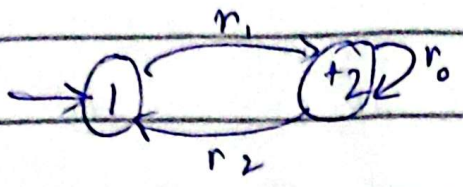
(2)



(3)



- It make sence that there are 2 loops at final state



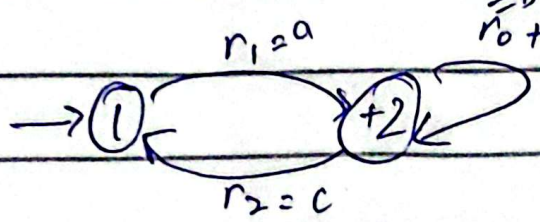
→ final state have 2 loops

① r_0

② $r_2 r_1$

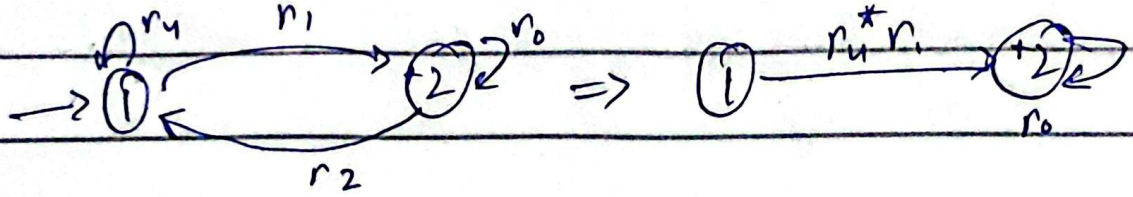
- From previous rules we can convert 2 loops into one :

$r_1 r_0^* (r_2 r_1 r_0^*)^*$



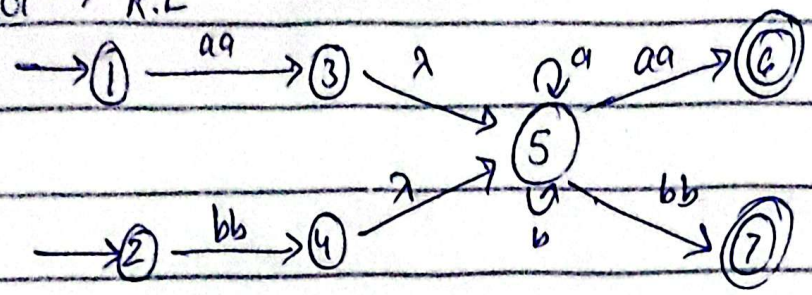
R.E → $ab^*(cab^*)^*$

- If both final and initial states have loops



$r_4^* r_1 (r_0^* r_2 r_4^* r_1)^* \Rightarrow$

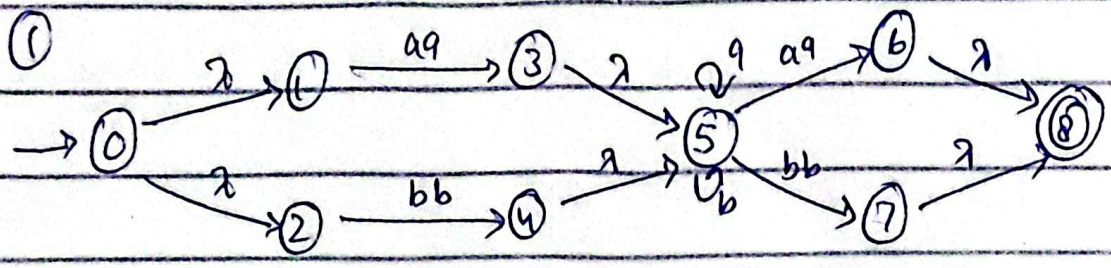
(1) TG → R.E



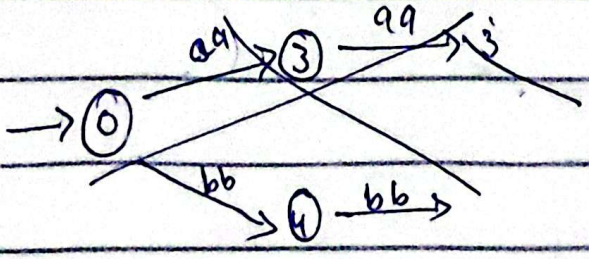
Step 1

Remove initial and final states

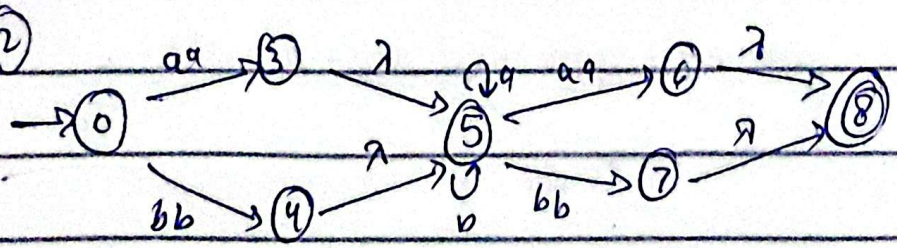
(1)



(2)

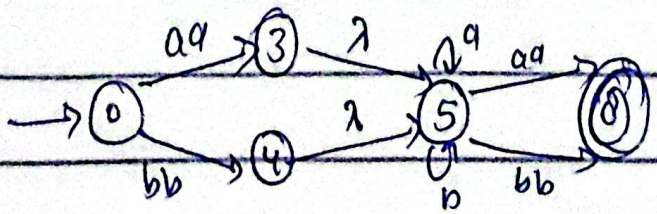


(2)

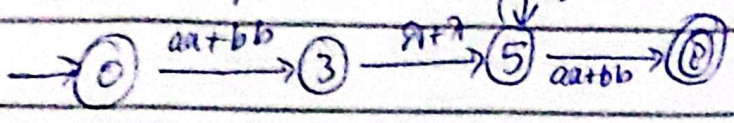


~~Step 2~~ (3)

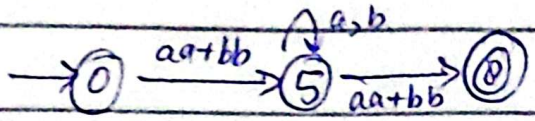
Remove



Step 2 Combine paths

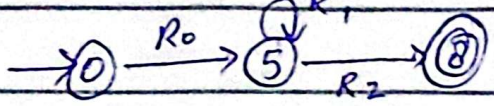


Step 3 Concatinate with λ (Merge λ)

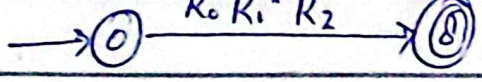


Step 4

Lets say $aa+bb = R_0$, $\overset{OR}{\underset{a,b}{a+b}} = R_1$, $aa+bb = R_2$

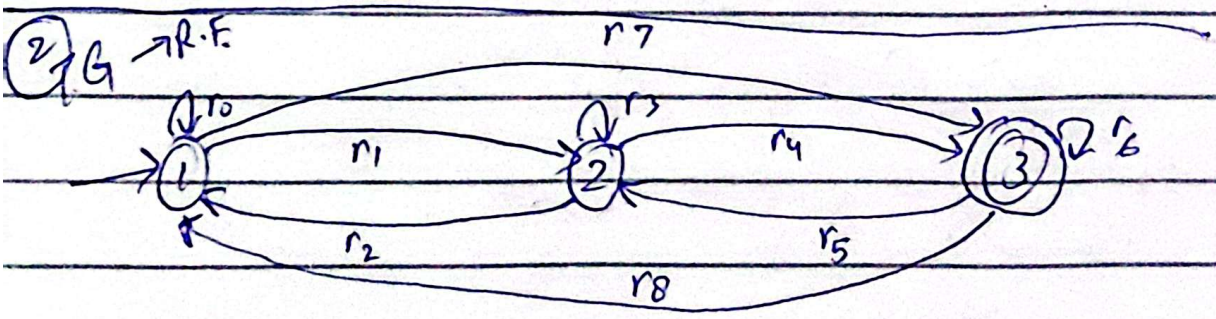


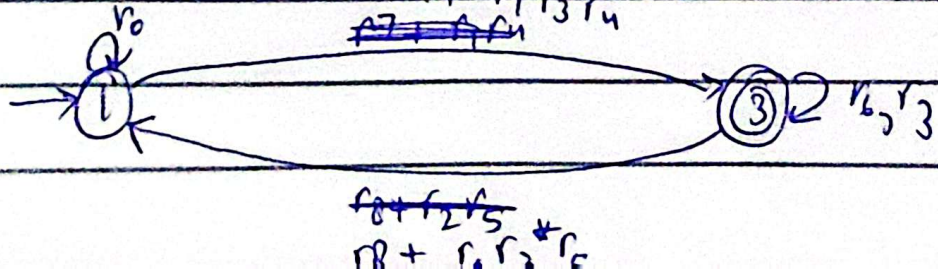
Step 5



This is our regular expression: $R_0R_1^*R_2$

$R_0R_1^*R_2 = (aa+bb)(a+b)^*(aa+bb)$



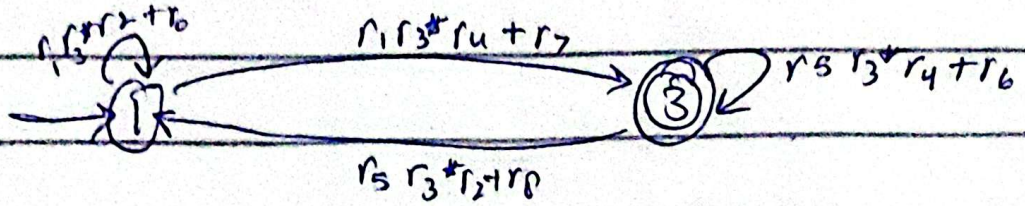


② T.G1 → R.E

Remove ②

States/Path →	Path	R.E
1 – 2 – 1	1 – 1	$r_1 r_3^* r_2$
1 – 2 – 3	1 – 3	$r_1 r_3^* r_4$
3 – 2 – 3	3 – 3	$r_5 r_3^* r_4$
3 – 2 – 1	3 – 1	$r_5 r_3^* r_2$

loops



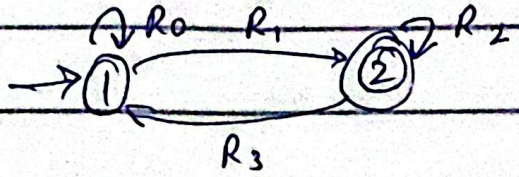
Lets say

$r_1 r_3^* r_2 + r_0 \rightarrow R_0$

$r_1 r_3^* r_4 + r_7 \rightarrow R_1$

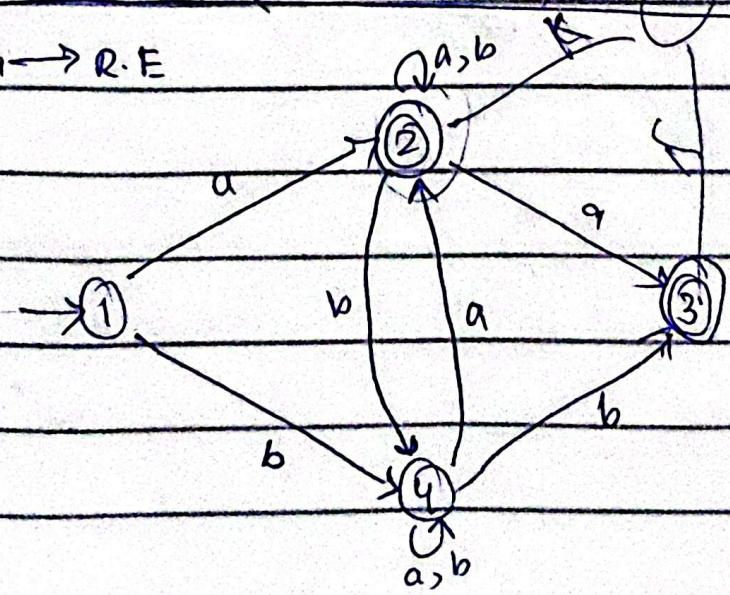
$r_5 r_3^* r_4 + r_6 \rightarrow R_2$

$r_5 r_3^* r_2 + r_8 \rightarrow R_3$



$R.E = R_0^* R_1 (R_2 + R_3 R_0^* R_1)^*$

③ T.G1 → R.E



Tuesday

Bootstrap

→ How to cu

↳ Always

→ After

→ Just ad

and all a

→ <script

to add bre

tag in <head

<div clas

<div

<

<

<div c

<

<d

<d

<div

<

<

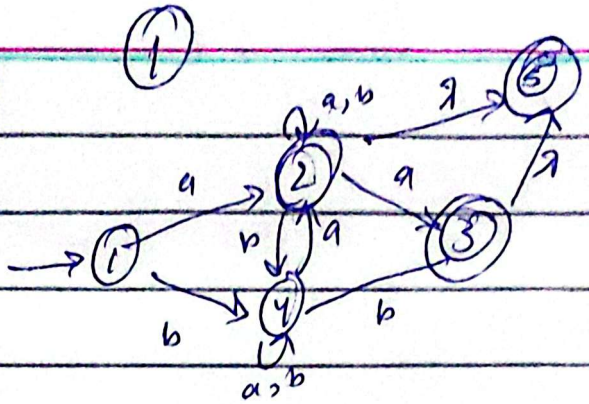
<d

<d

<d

1 — 2 — 3

1 — 2 — 4 — 3



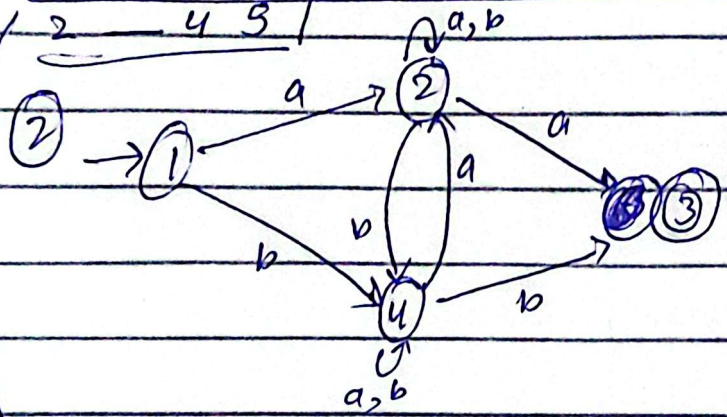
3

1 — 4 — 3

1 — 4 — 2

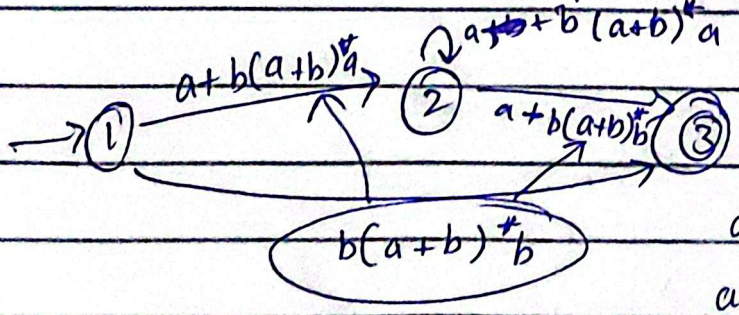
2 — 4 — 2

2 — 4 5



3

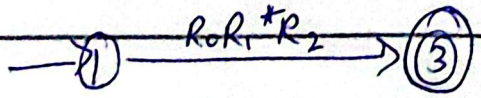
1 — 4 — 3	1 — 3	$b(a+b)^*b$
1 — 4 — 2	1 — 2	$b(a+b)^*a$
2 — 4 — 2	2 — 2	$b(a+b)^*a$
2 — 4 — 3	2 — 3	



$a+b(a+b)^*a$ — R_0

$a+b(a+b)^*a$ — R_1

$a+b(a+b)^*b$ — R_2



(III) Kleens Theorm

we can
R.E → Concatinate
↳ Closure $(a^*)(a^+)$
↳ Union

3- RE → FA (DFA)

L1: R1 : $(a+b)^*b$ – string ends with b

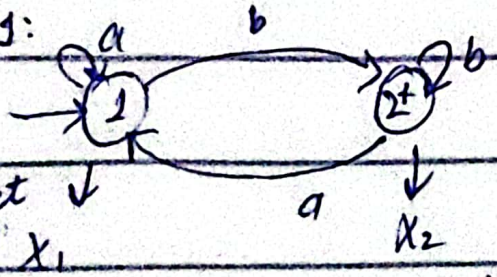
L2 : R2 : $(a+b)^*aa(a+b)^*$ – string contain aa

→ $L1 \cup L2 \Rightarrow R1 \cup R2 \Rightarrow (a+b)^*b + (a+b)^*aa(a+b)^*$

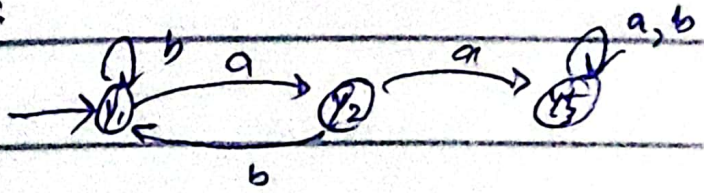
(+ : choice operator is used for union)

DFA →

L1:

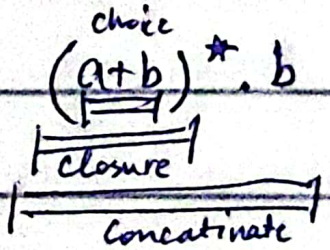


L2 :

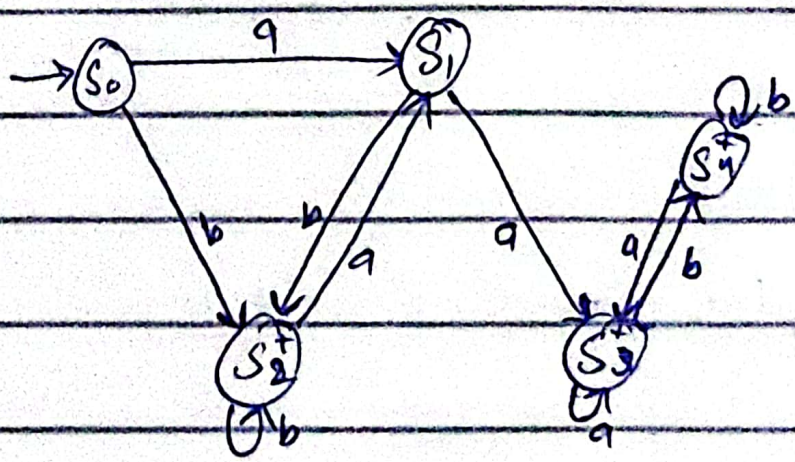


Y_3, X_2 → final states

DFA's Union



State	a	b
S_0 (X_1, Y_1)	S_1 (X_1, Y_2)	S_2 (X_2, Y_1)
S_1 (X_1, Y_2)	S_3 (X_1, Y_3)	S_2 (X_2, Y_1)
S_2 (X_2, Y_1)	S_1 (X_1, Y_2)	S_2 (X_2, Y_1)
S_3 (X_1, Y_3)	S_3 (X_1, Y_3)	S_4 (X_2, Y_3)
S_4 (X_2, Y_3)	S_3 (X_1, Y_3)	S_4 (X_2, Y_3)



class = modal-dialog ~~role~~ = model-md.

class = modal-contents ← contains

- Header ⇒ class = modal-header
- Body ⇒ class = modal-body
- Footer ⇒ class = modal-footer

Tabs

class = "nav nav-pills

Pills - To show data on front-end using single page (blue color buttons)

↳ data-toggle = "pill" use id = "menu1"

↓

must have a content dive

class = tab-content use the same id = "menu1"

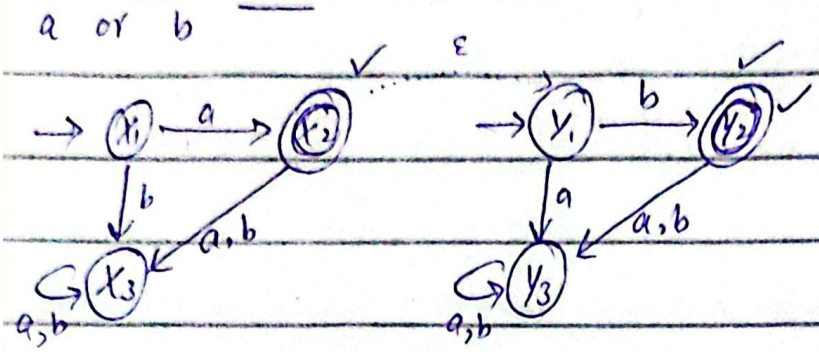
Bootstrap Forms →

```
<form class = form-group>
    <input type = text class = Form-control>
    :
    rest is same
```

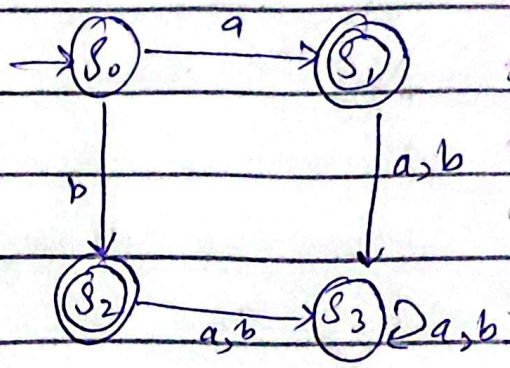
RE → DFA (FA) $\Sigma = \{a, b\}$ ① Union

R.E => a+b ① Union ② Concatination

a or b ③ Closure



States	a	b
(X_1, Y_1) S_0	$S_1(X_2, Y_1)$	$S_2(X_3, Y_2)$
$S_1(X_2, Y_3)$ +	$S_3(X_3, Y_3)$	$S_3(X_3, Y_3)$
$S_2(X_3, Y_2)$	$S_3(X_3, Y_3)$	$S_3(X_3, Y_3)$
$S_3(X_3, Y_3)$	$S_3(X_3, Y_3)$	$S_3(X_3, Y_3)$

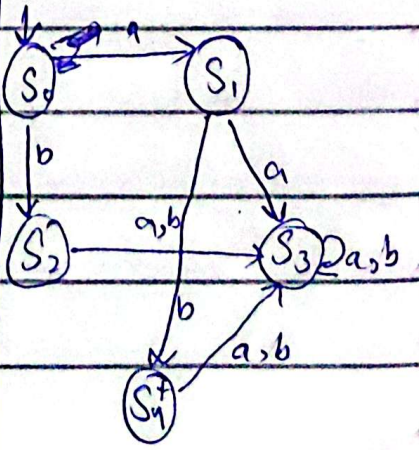


R.E => a.b ② Concatination

=> Automata of first expression leads to automata of last expression, how?

↳ Link / Include the final state of first automata with initial state of 2nd

States	a	b
$S_0(X_1)$	$S_1(X_2, Y_1)$	$S_2(X_3)$
$S_1(X_2, Y_1)$	$S_3(X_3, Y_3)$	$S_4(X_3, Y_2)$
$S_2(X_3)$	$S_3(X_3 Y_3)$	S_3
$S_3(X_3, Y_3)$	S_3	S_3
$S_4(X_3, Y_2)$	$S_3(X_3, Y_3)$	$S_3(X_3, Y_3)$



$a^+ = a.a^*$

R.E => a^* (3) Closure {a, aaa, b, bbb, abb, aab ...}

L1: odd length of string R.E:

L2: contain atleast one a R.E: ~~$a^*(a+b)^*a^*$~~

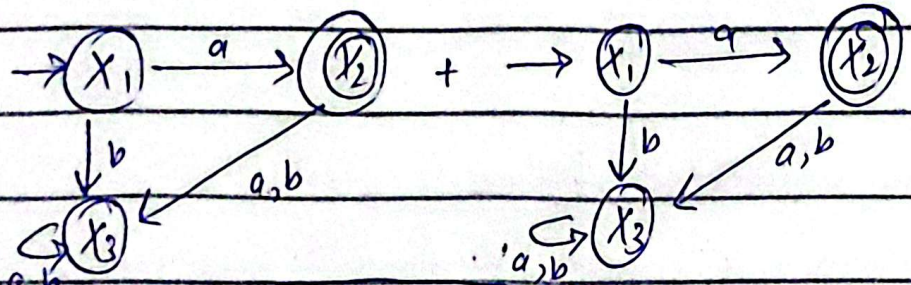
L1.L2

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

RE_2: $a(a+b)^* + (a+b)^*a \Rightarrow (a+b)^*a(a+b)^*$

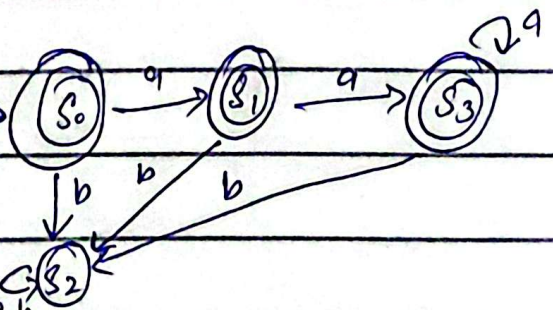
→ Choose the initial state also a final state

→ rest is same as concatination



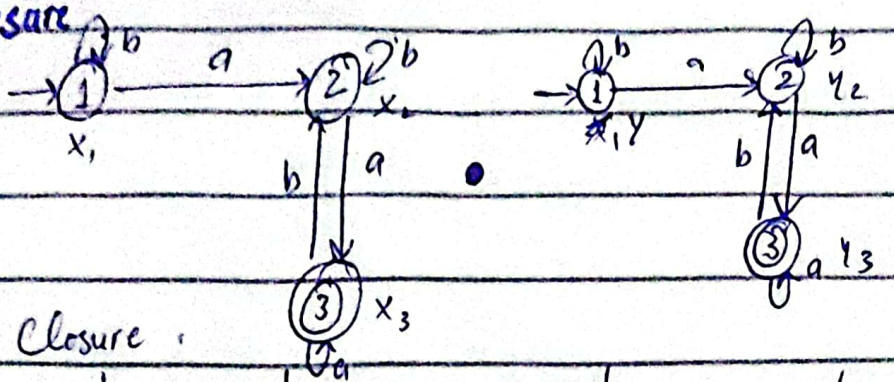
Don't repeat S_0

state	a	b
→ $S_0(X_1)$	$S_1(X_2,X_1)$	$S_2(X_3)$
+ $S_1(X_2,X_1)$	$S_3(X_3,X_2,X_1)$	$S_2(X_3)$
$S_2(X_3)$	$S_2(X_3)$	$S_2(X_3)$
+ $S_3(X_3,X_2,X_1)$	S_3	S_2



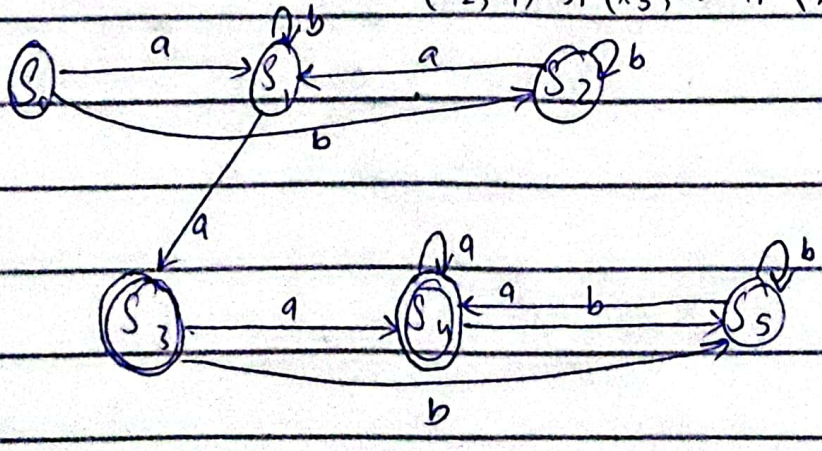
Exception: if initial state have a loop or a state transition takes back to initial state (S_0) then don't assign it S_0 again cuz initial state must be only one. Assign it a different name.

Closure

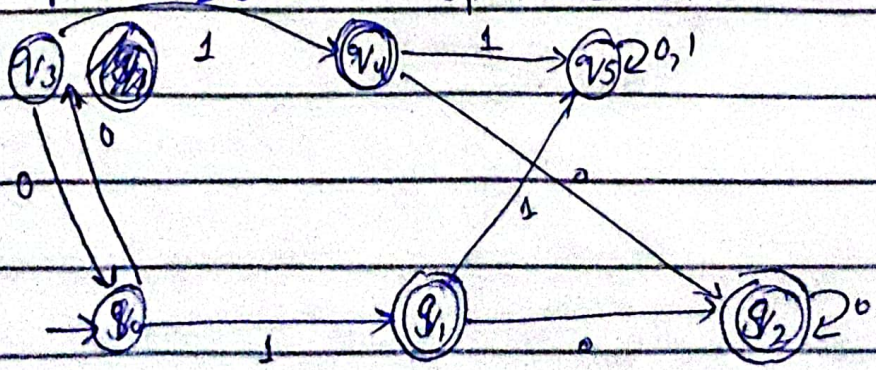


① Closure :

state	a	b
~~$(X_1, Y_1) S_0$~~	~~$(X_2, Y_2) S_1$~~	~~(X_1, Y_1)~~
$+(X_1) S_0$	$(X_2, X_1) S_1$	$(X_1) S_2$
$(X_2) S_1$	$(X_3) S_3$	$(X_2) S_1$
$(X_1) S_2$	$(X_2) S_1$	$(X_1) S_2$
$+(X_3) S_3$	$(X_3, X_2) S_4$	$(X_2, X_1) S_5$
$+(X_3, X_2, X_1) S_4$	$(X_3, X_2) S_4$	$(X_2, X_1) S_5$
$(X_2, X_1) S_5$	$(X_3, X_2) S_4$	$(X_2, X_1) S_5$



DFA Minimization – Optimize DFA



G_1 (Non-Final) G_2 (Final states)

$\{q_0, q_3, q_5\}$ $\{q_1, q_2, q_4\}$

0-Equivalance

	0	1	
q_0	G_1	G_2	Same
q_3	G_1	G_2	
q_5	G_1	G_1	Different

	0	1	
q_1	G_2	G_1	Same
q_2	G_2	G_1	
q_4	G_2	G_1	

- Split the difference

G_1 $\{q_0, q_3\}$ G_2 $\{q_5\}$ G_3 $\{q_1, q_2, q_4\}$

1-s Equivalance

	0	1
q_5	G_2	G_2

G_1

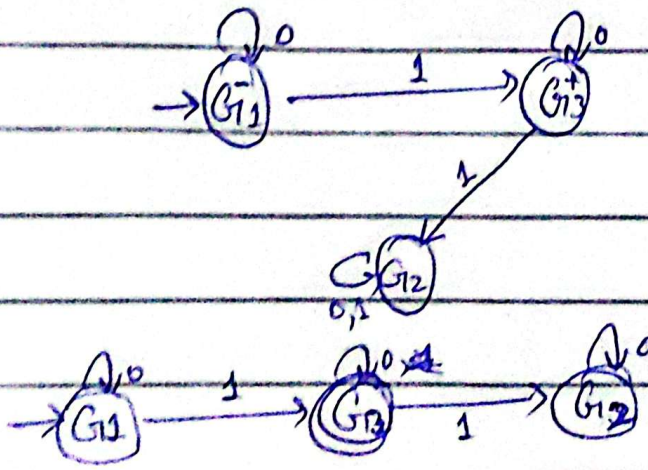
	0	1	
q_0	G_1	G_3	Same
q_3	G_1	G_3	

G_2 – no need cuz only one state

G_3

	0	1	
q_1	G_3	G_2	Same
q_2	G_3	G_2	
q_4	G_3	G_2	

- No need to further split cuz there is no more difference



If G_3 further divides then those 2 states will be final states

Finite Automata with output

(1) Moore's Machine $\{\Sigma, Q, v_0, F, \delta\}$

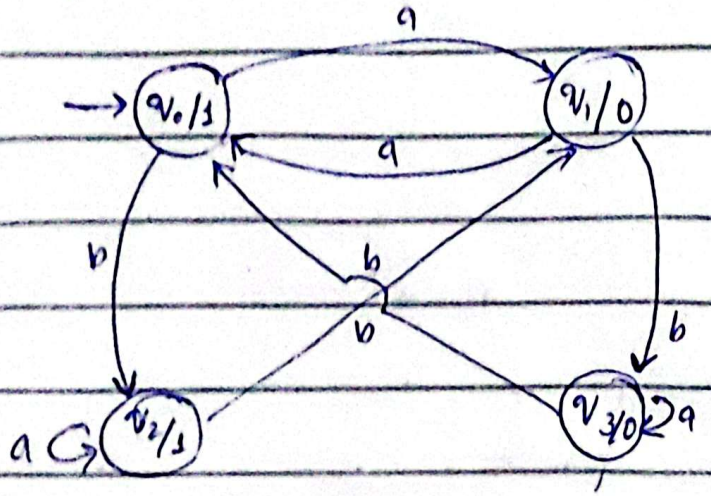
(2) Melay's Machine — with output

Output — Without final st. $\{\Sigma, Q, v, \Gamma, \delta\}$

1 - Moor's Machine

$\Sigma = \{a, b\}$, $Q = \{v_0, v_1, v_2, v_3\}$ $v_0 \Rightarrow$ Initial

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

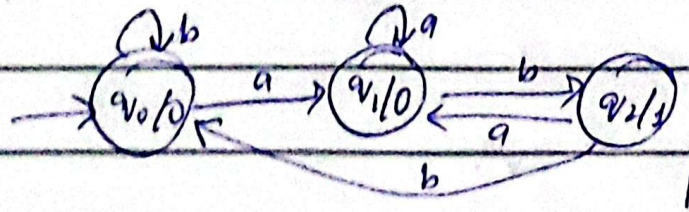


	a	b	output
v_0	v_1	v_2	1
v_1	v_0	v_3	0
v_2	v_2	v_1	1
v_3	v_2	v_0	0

Input : ~~abbab~~ (Initial st.) | a b b a b

Output : ~~10010~~ 1 (output) | 0 0 1 0 0

Example # 1



Identify "ab" as substring and ~~opto~~ the output associated = 1

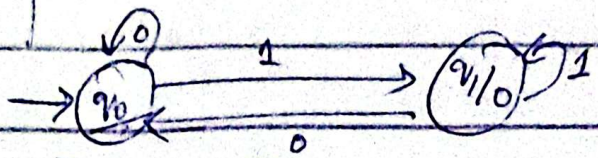
Input	b a a b b
Output 0	0 0 0 1 0

Input	a a b b a b
Output 0	0 0 1 0 0 1

Moor's Machine $\Rightarrow$ Output on states

Example# 2 Find the 1's compliment of input string.

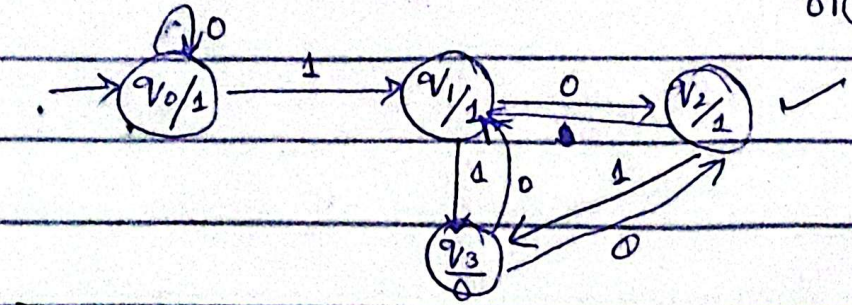
Input	10 10 11
Output	0 1 0 1 0 0

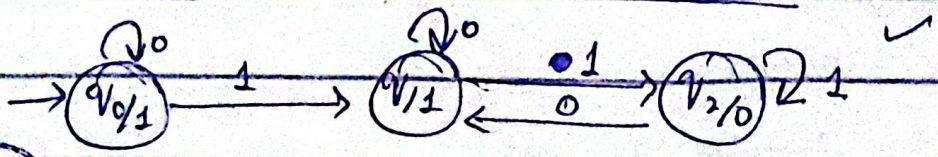


Ex# 3 Find 2's compliment of input string

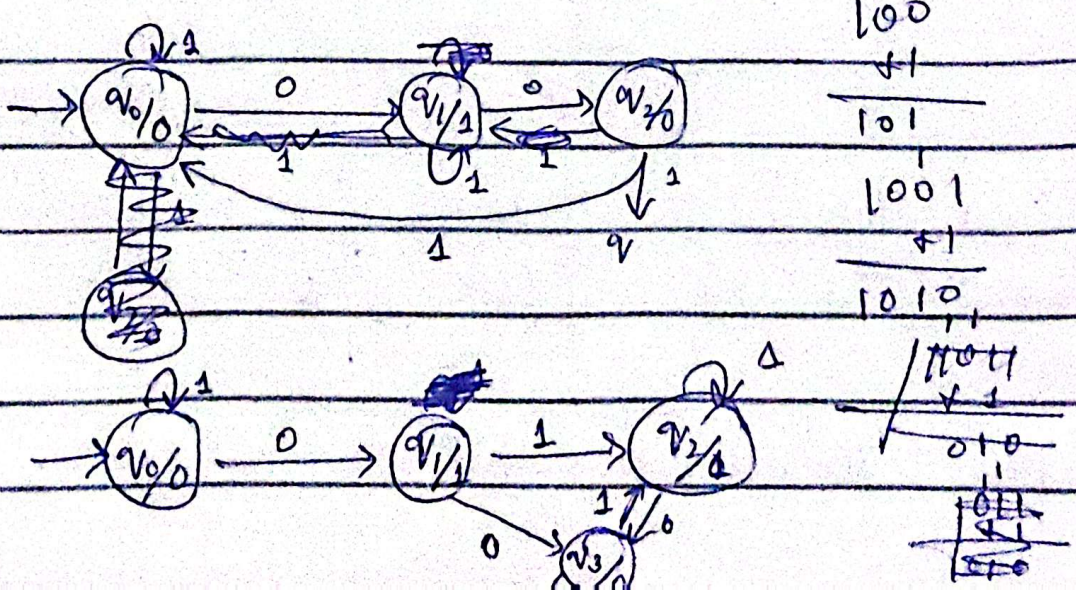
10 100 ← R

01 1 00





Ex# 4 Add 1 in any binary string with same input and output length

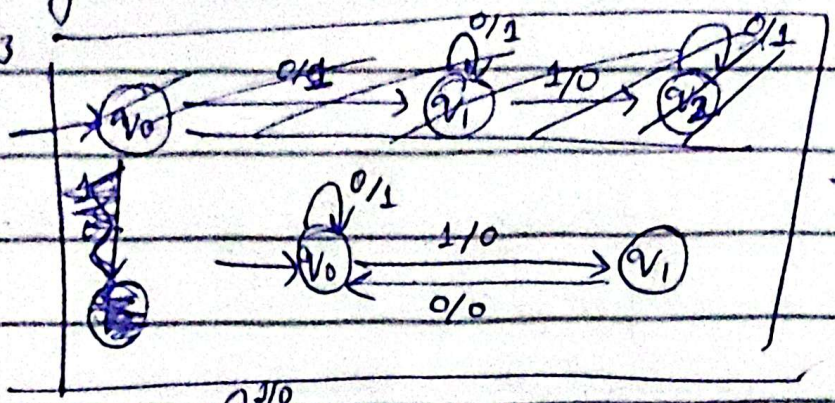


100
+1
101
1001
+1
1010

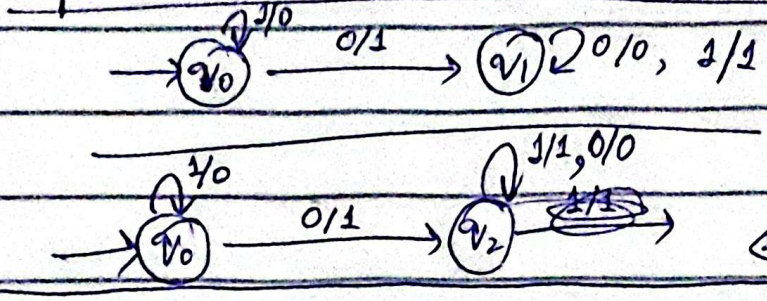
01010
+ 1
01

2- Mealy's Machine : Output on transitions

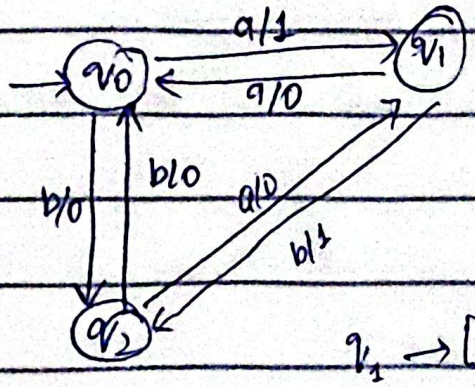
Ex#3



0	1
+1	
100..	



Melay to Moore



• Split the state with different output

	a	b
q_0	q_2, 1	q_2, 0
q_1	q_0, 0	q_2, 1
q_2	q_1, 0	q_0, 0

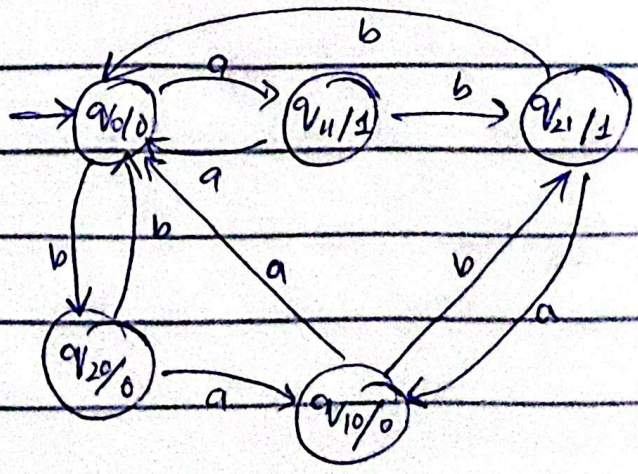
split

split

$q_1 \rightarrow [q_{10}, q_{11}]$

$q_2 \rightarrow [q_{20}, q_{21}]$

	a	b
$0 \leftarrow q_0$	q_{11}	q_{20}
$0 \leftarrow q_{10}$	q_0	q_{21}
$1 \leftarrow q_{11}$	q_0	q_{21}
$0 \leftarrow q_{20}$	q_{10}	q_0
$1 \leftarrow q_{21}$	q_{10}	q_0



- Repeat the first transation into the same state

Input a a b a

Mealy	1	0	0	0
Moore	0/1	0	0	0

ToA

1. Regular Expression
2. DFA
3. NFA
3. TG
4. Kleens Theorm ✓

   (i) F.A → T.G (ii) T.G → R.E (iii) R.E → F.A(DFA)

5. DFA Minimization
6. Moore's Machine
7. Melay's Machine
8. FA with Output

Kleenes Theorm

Any language that can be defined by

1. Regular expression (R.E)
2. Finite automaton (F.A)
3. Transition graph (T.G)

can be defined by all three methods.

FA with Output

Moore's → Melay's Machine

- Regular language

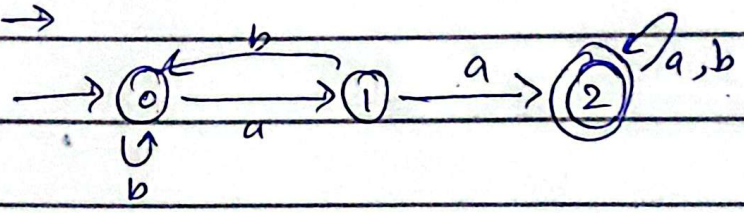
Union, Concatenation, Closure, Intersection (4)

L1 :- aa => {aa, aab, baa, aaa, ...}

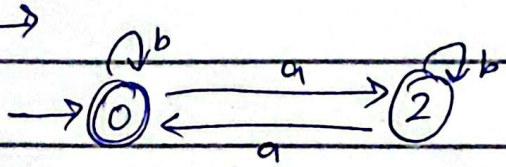
L2 :- Even no. of a's => {λ, b, bb, ..., aab, baa..}

$L1 \cap L2 = (L1' \cup L2')'$

L1 →



L2 →

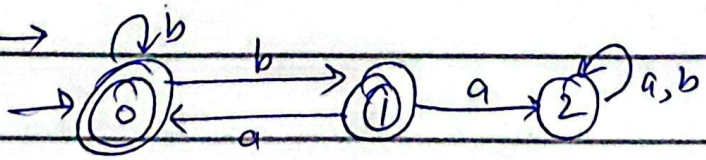


⇒ Compliment of language = negate the language

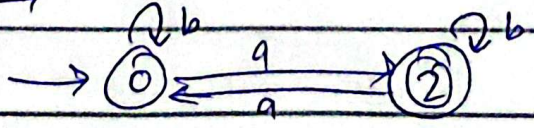
↓

change final to non-final & vice versa

L1' →



L2' →

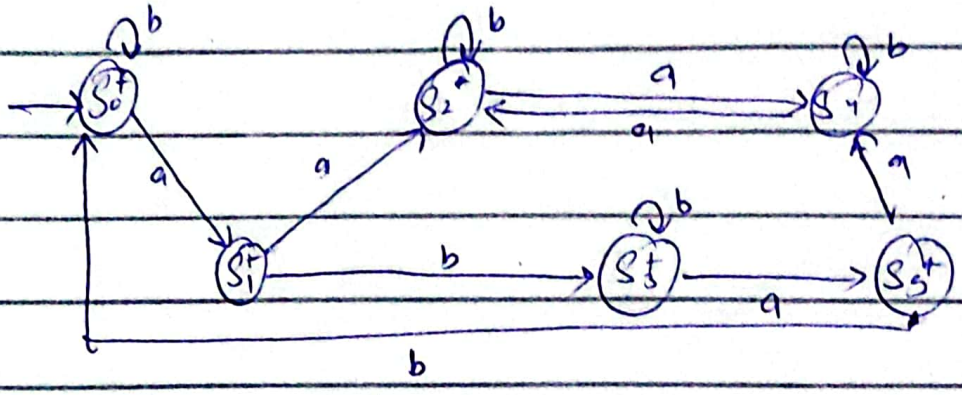


Now $L1' \cup L2'$

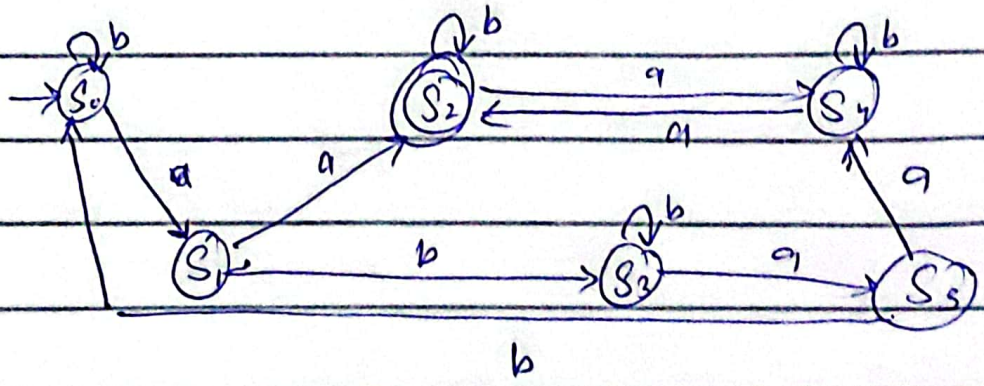
$L_1' \cup L_2' \rightarrow$

State	a	b
$S_0(x_0, y_1)$	$S_1(x_1, y_2)$	$S_0(x_0, y_1)$
$S_1(x_1, y_2)$	$S_2(x_2, y_1)$	$S_3(x_0, y_2)$
$S_2(x_2, y_1)$	$S_4(x_2, y_2)$	$S_2(x_2, y_1)$
$S_3(x_0, y_2)$	$S_5(x_1, y_1)$	$S_3(x_0, y_2)$
$S_4(x_2, y_2)$	$S_2(x_2, y_1)$	$S_4(x_2, y_2)$
$S_5(x_1, y_1)$	$S_4(x_2, y_2)$	$S_0(x_2, y_1)$

$L_1' \cup L_2' \rightarrow$



$(L_1' \cup L_2')'$



Monda

PHP

- hype
- exe

(Simp
Databa

① Server

② DB =

③ User

④ Pass

$ link=
mcqli_
mysql

Xampp
- mys
- Apa

Tuesday ~~Monday~~ ToA 19-3

Language of Palindrome $\Sigma = \{a, b\}$

→ Finite Autometa produces infinitely many string which is part of regular language.

Finite Autometa → Have finite states but they must have repetition (loops) to produce regular expression string.

→ Finit Autometa which have finite state ~~But~~ with repetition but produce string which are not part of rglar language are non-regular language

Pumping Lemma : Used to prove non-regular lang.

1. Suppose: Language (L) is regular language.
2. The finite automata of language consists of N no. of finite states.
3. Suppose string $w \in L$ where $|w| > N$
4. Divide string w in a substring $xyz \Rightarrow w = xyz$

3 parts

5. $|y| \neq 0$
6. $|x| + |y| \leq N$

$w = xy^i z \Rightarrow i \geq 1$

if $w \in L \rightarrow L$ is regular

otherwise it is non-regular

Palindrome | Forceful Break Test | ✓

{a, b, aa, bb, aba, bab, ...}

N = 5

$y^1 \rightarrow$ $w = \underbrace{baa}_{x}\underbrace{b}_{y}\underbrace{aab}_{z}$ | $w = xy^i z$

$y^2 \rightarrow$ $w = \underbrace{baa}_{x}\underbrace{bab}_{y^1}\underbrace{aab}_{y^2 z}$ | $w = xy^2 z$

$y^3 \rightarrow$ $w = \underbrace{baa}_{x}\underbrace{bab}_{y^1}\underbrace{aba}_{y^2}\underbrace{ab}_{y^3 z}$ | $w = xy^3 z$

$y_1 \rightarrow$ $w = \underbrace{baa}_{x}\underbrace{baab}_{y^1 z}$ | $w = xy^1 z$

$y_2 \rightarrow$ $w = \underbrace{baa}_{x}\underbrace{baa}_{y^1}\underbrace{baab}_{y^2 z}$ | $w = xy^2 z$

Non-regular

Ex#1 $a^n b^n$ → Non regular $n \geq 1$

$\{ab, aab, abb, aaa, \ldots\}$

$N = 4$

$w = \underbrace{aa}_{x}\underbrace{ab}_{y}\underbrace{b}_{z}$ → $x+y \leq N$

$xy^1z = \underbrace{aa}_{x}\underbrace{ab}_{y^1}\underbrace{b}_{z}$ | $xy^2z = \underbrace{aa}_{x}\underbrace{ab}_{y^1}\underbrace{ab}_{y^2}\underbrace{b}_{z}$ ∉ R.L

Myhill Nerode Theorem: used to prove regular lang.

- Language L is regular language over Σ
- String contains even no. of a's & b's

Σ^*

C1 → even a's and even b's

C2 → even a's and odd b's

C3 → odd a's and even b's

C4 → odd a's and odd b's

} Pick string x, y

- Pick string x and y from same class
- Pick another string z from different class
- If xz and yz both string belongs to the language classes then it is regular otherwise it is non-regular.

C3 [x = bab | xz = babab
     y = abb | yz = abbab

C4 − z = ab

Regular language

Palindrome

C1 = Palindrome C2 = Non-Palindrome

x = aba , y = aa , z = ba

xz yz

ababa aaba } non-regular

palin non

C1 C2

Decideability

→ Effectively Solvable - algorithm exists which solves the issue in finite steps

→ Decision Procedure - Yes/No

↳ for whether the prob. can be solved

→ if prob. has Decision Procedure ; we call it Decidability Problem

R.E

① Remove every kleene star (*) operator
② Remove every kleene plus (+) operator
} exponent

③ Remove everything on right hand side of union operator

④ Remaining will only be paranthesis or concat.

eg

$\Rightarrow (a+b)^*(abb+\lambda)(a^+ + aa)^*$

① $(a+b)(abb+\lambda)(a^+ + aa)$

② $(a+b)(abb+\lambda)(a^* + aa)$

③ $(a)(abb)(a) \Rightarrow \underbrace{aabba}$

If empty then

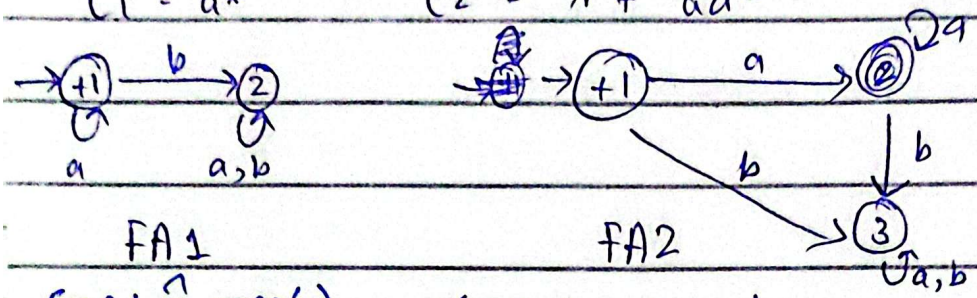
Intro
Cohen

FA

① If any path to final state; it accepts strings

E.g (Hometask)

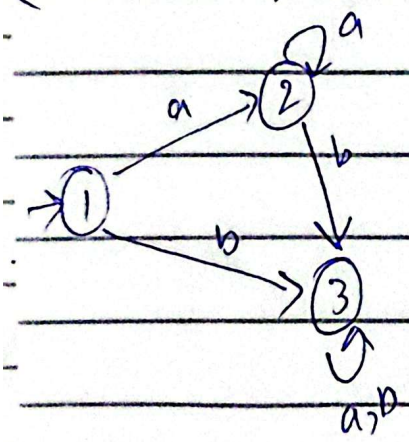
$L1 = a^*$ $L2 = \lambda + aa^*$



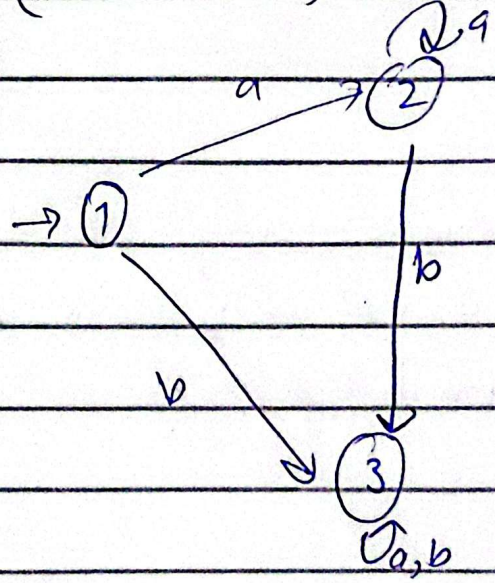
FA1 FA2

$(FA1 \cup FA2') = (FA1' \cap FA2)'$

$(FA1' \cup FA)$



$(FA1 \cup FA2')'$



Context Free Grammer

① a's & b's Even No.

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

~~$S \to aaA \mid bbA \mid \lambda$~~

~~$A \to abA \mid baA \mid \lambda$~~

$S \to aaS \mid bbS \mid XSX \mid \lambda \mid SS$

$X \to ab \mid ba$

abab aababa

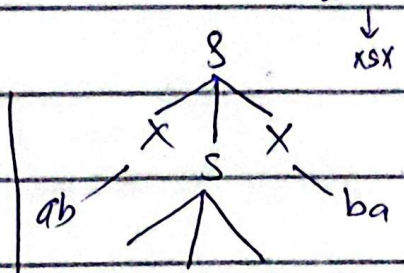
ab λ ab

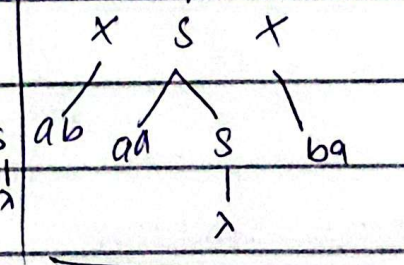
XSX

~~XSX~~

aaS

XSX





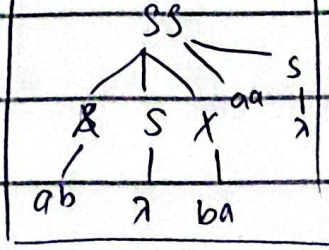
② $a^*\ b^*$

- all a's before b's and null

~~$S \to aaS \mid bbS \mid \lambda$~~

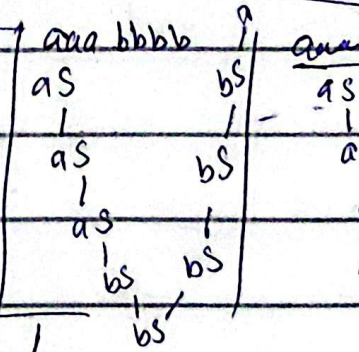
abbaaa

SS



- $S \to aS \mid bS \mid \lambda$ → can produce ba which is invalid

aaa bbbb



aaaaa

① $S \to aS \mid bX \mid \lambda$

$X \to bX \mid \lambda$

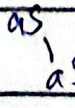
② $S \to AB$

$A \to aA \mid \lambda$

$B \to bB \mid \lambda$

③ $S \to aS \mid Sb \mid \lambda$

→ aab



→ aab

AB

aA bB

aaAbB

~~aaAbB~~

aaλbλ

aab

Ambigious Grammer

- Derivation Left and Right
  - ↳ more than 1

- id + id * id

$E \to E+E \mid E*E \mid$

E+E

id + E * E

id + id * id

Left

E * E

E + E

id + id

id

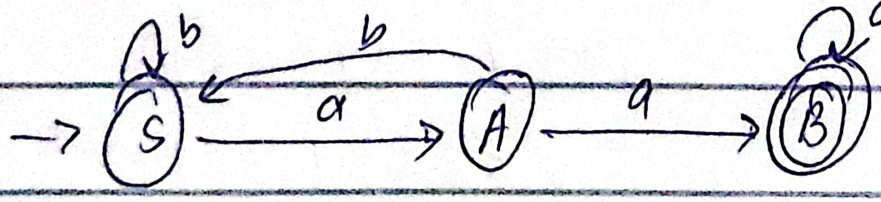
Right

Tuesday ToA 2-4

Regular Grammar

- Context Free Grammer of regular expressions
- FA → CFG

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



Regular Grammer

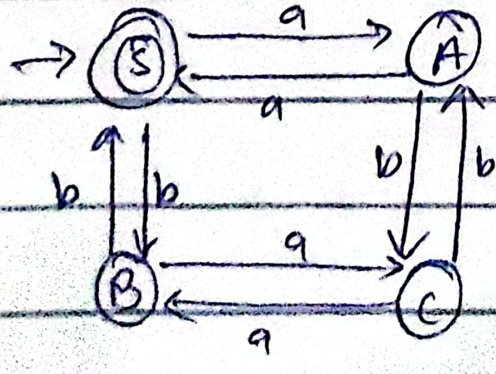
$S \rightarrow aA \mid bS$

$A \rightarrow aB \mid bS$

$B \rightarrow aB \mid bB \mid \lambda$

R.E : Even no. of a's and even no. of b's

Final state → λ



Regular Grammer

$S \rightarrow aA \mid bB \mid \lambda$

$A \rightarrow aS \mid bC$

$B \rightarrow bS \mid aC$

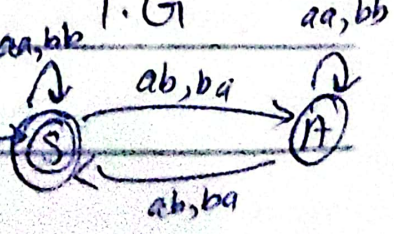
$C \rightarrow aB \mid bA$

→ CFG of R.L has certain pattern:

RG → Semiword | word

Semiword → (terminal)*

T.G



$S \rightarrow aaS \mid bbS \mid abA \mid baA \mid \lambda$

$A \rightarrow aaA \mid bbA \mid abS \mid baS$

babaaa

RG1 → semiword | word

word → terminal only

semiword → (terminal)* nonterminal

CHOMSKY'S Normal Form (CNF)

① Remove Null and Nullable Production

② Remove unit production.

Non-terminal → Exactly Two non-t., Single terminal

Example:

S → ab | bA | λ (Null)	S → ab | bA | λ (Null)
A → aSa | bS | ab (Nullable)	A → aSa | AbS | ab | S (Nullable)
① S → ab | bA	S → ab | bA | b
A → aSa | bS | ab | aa | b	A → aSa | AbS | ab | ~~bS~~ | aa | Ab | b

Example 2:

S → Xa | YY | aX | ZYX	S → Xa | YY | aX | ZYX | ZX | Y
X → Za | bZ | ZZ | Yb	X → Za | bZ | ZZ | Yb | b
Y → Ya | XY | λ	Y → Ya | XY | a | X
Z → aX | YYY	Z → aX | YYY | YY | Y

S → Xa | YY | aX | ZYX | ZX | YX | a | Z | XZ | ZY | X | Y

X → Za | bZ | ZZ | Yb | a | b | Z

Y → Ya | XY | a | Y | X

Z → aX | YYY | YY | Y | a

② S → ab | bb

A → S | a

S → ab | bb

A → ab | bb | a

② Cyclic :-

S → A | bb → B | b | bb → S | a | b | bb → a | b | bb

A → B | b → S | a | b → a | b | bb

B → S | a → a | b | bb

Convert to CNF

$S \to aA \mid bB \mid \lambda \to aA \mid bB$	① $S \to aA \mid bB$
$A \to aa \mid SA$	$A \to aa \mid SA$
$B \to aS$	$B \to aS$

② $S \to aA \mid bB \to aA \mid bB \mid a \mid b$

$A \to aa \mid SA \mid \lambda \mid A$

$B \to aS \mid a$

③ $S \to aA \mid bB \mid a \mid b$

$A \to aa \mid SA \mid aA \mid bB \mid a \mid b \mid aS$

$B \to aS \mid a$

④ $S \to aA \mid bB \mid \lambda$ ← $S \to \lambda$

$A \to aa \mid SA$ (5)

$B \to aS$

(i) nullable null remove

$S \to aA \mid bB$

$A \to aa \mid SA \mid aa \mid A$

$B \to aS$

$S \to ab \mid bA \mid \lambda$

$A \to aSa \mid bS \mid ab$

(ii) $S \to ab \mid bA \mid$

$A \to aSa \mid bS \mid ab \mid aa \mid b \mid$

PushDown Automata (PDA)

① Input Tape ② States => {Accept, Reject, Start Reg}
③ Stack => {push, pop} ④ Σ => Input Alphabet

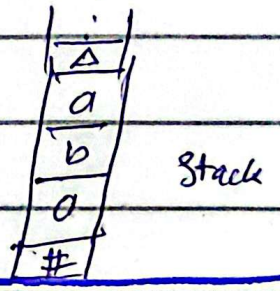
Memory Components

① **Tape** → behave like array
- Initially the tape is empty, have 'Δ' on indexes
- Can read only in forward direction

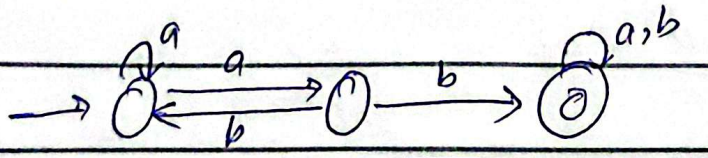
Tape example: 'aba'

a	b	a	Δ	...

③ **Stack** → same as stack
- all the properties of stack, =>

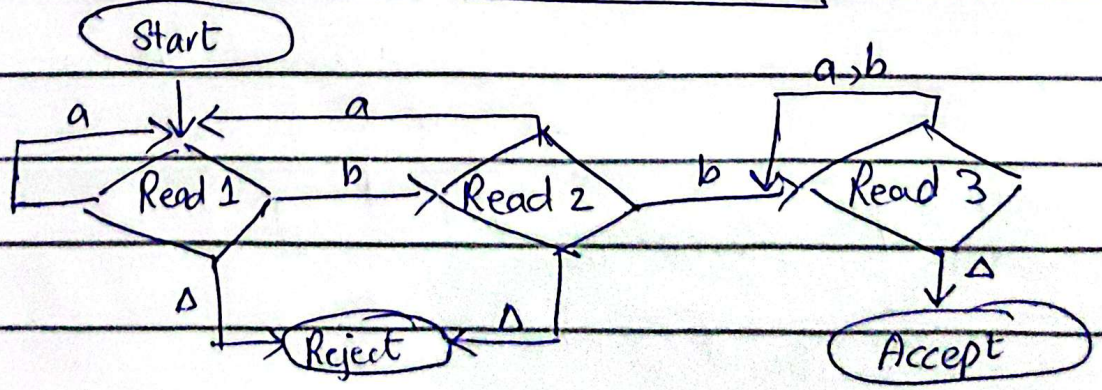


R.E: $(a+b)^*bb(a+b)^*$ → For R.L use Tape



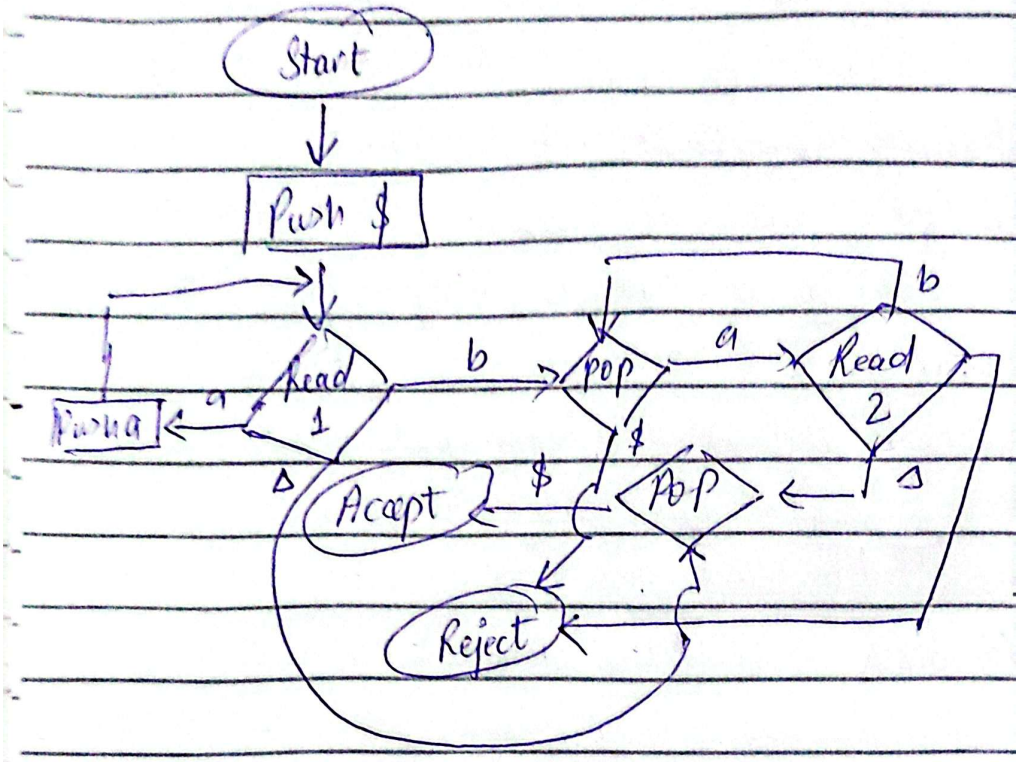
Tape:

a	b	b	a	Δ	...



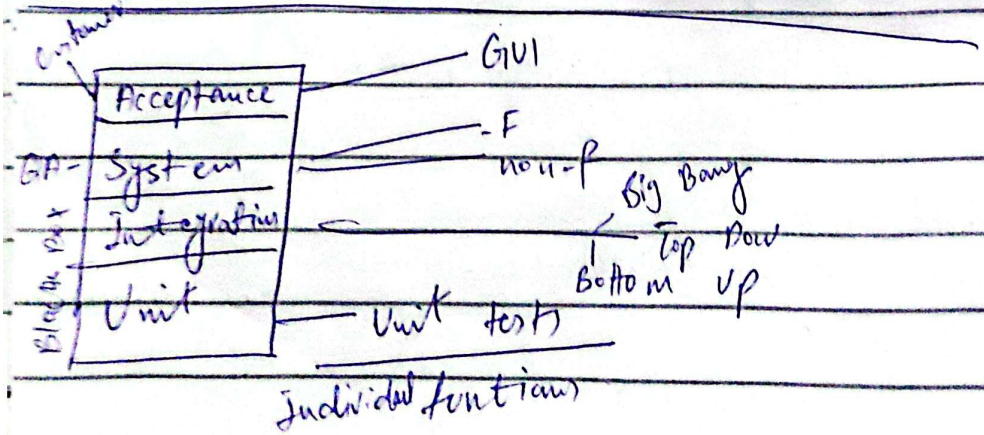
R.L = aaabbb R.L 'a'

$a^n b^n$ → For non-regular use stack



$a^n b^{2n}$ → Stack

R.L = aabbbb → Pop on 2 b's

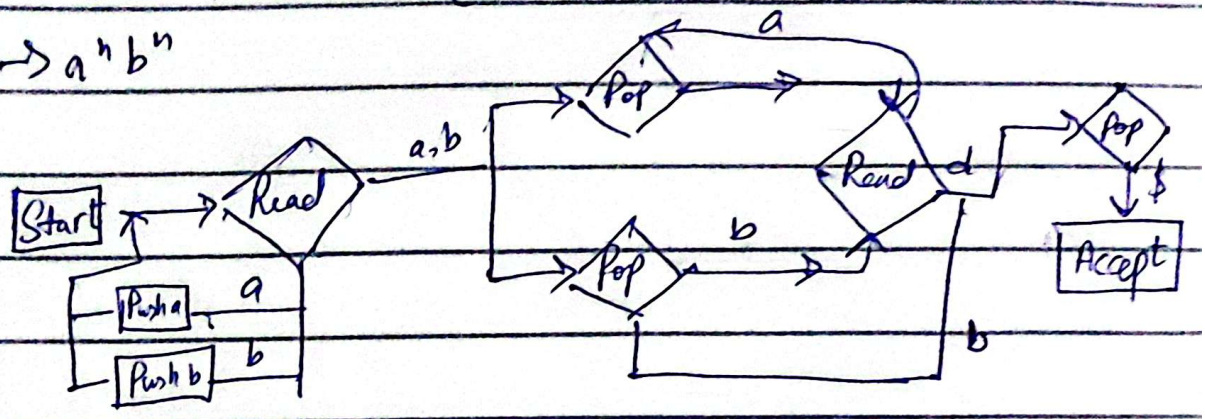


Online Lect#1

Monday TOA 22-4

Push Down Automata (PDA)

→ $a^n b^n$



→ bbaabb

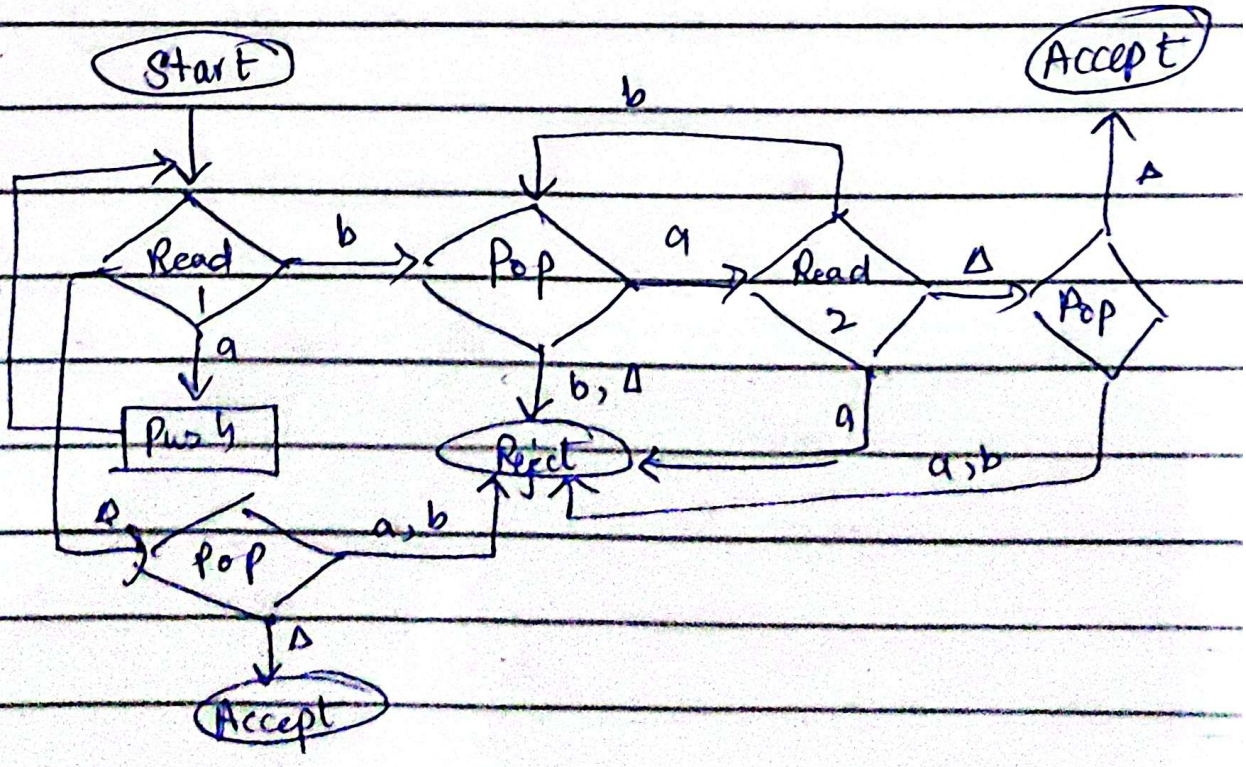
→ Only even length of string

bbaabb

push pop

→ can perform multiple options/ choices

- Can Push or Pop or as a Read choice

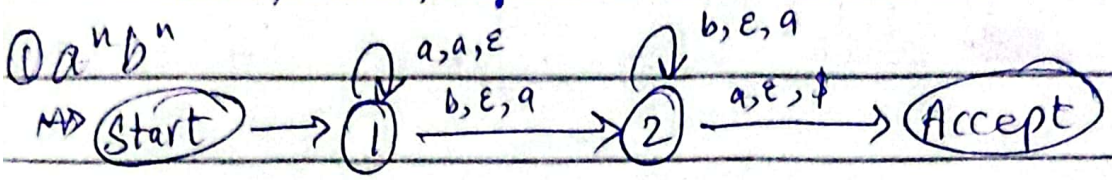


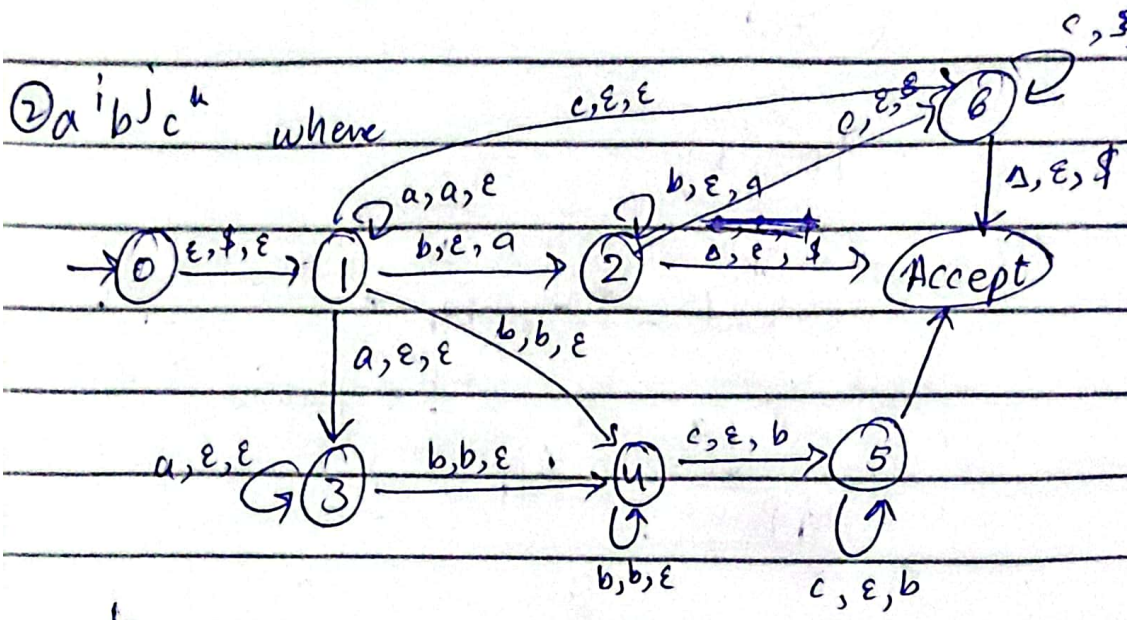
(Final2)

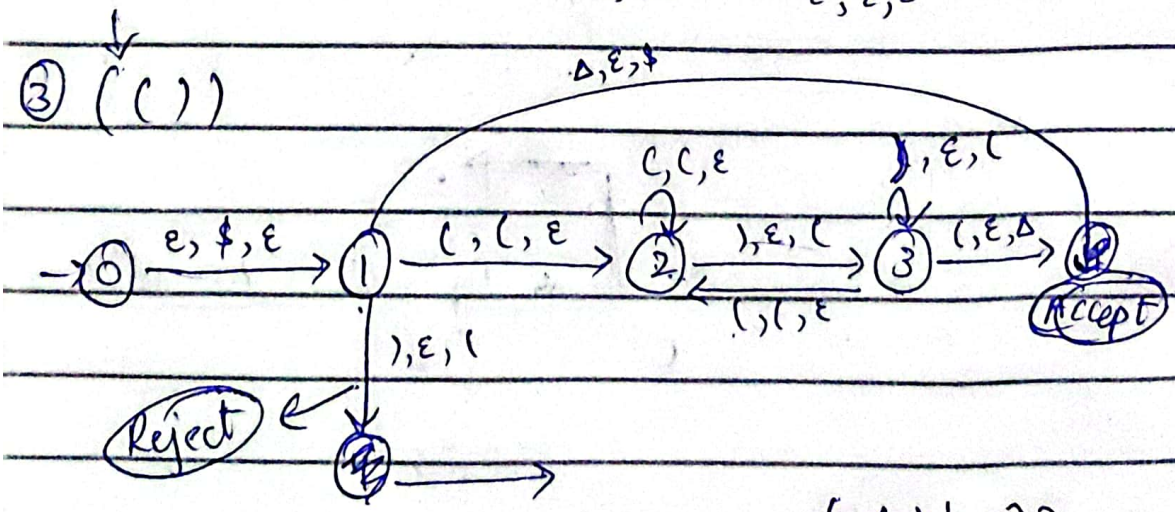
ε → Null, Nothing

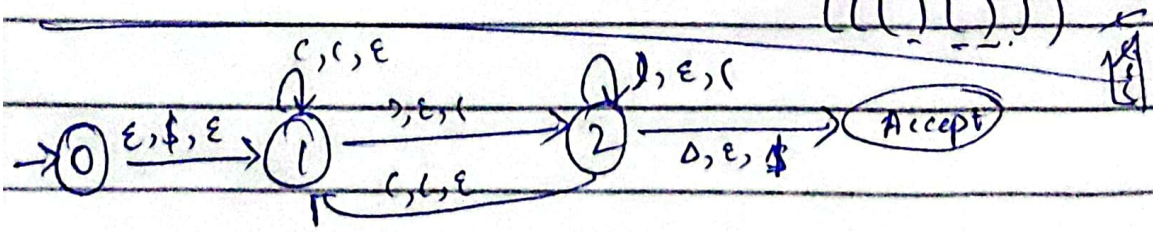
PDA - Easy way

Read, Push, Pop







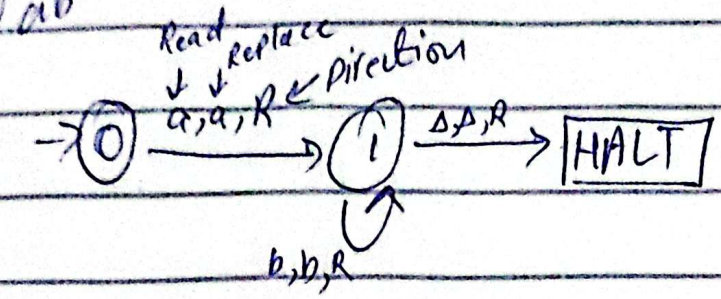


Turing Machine

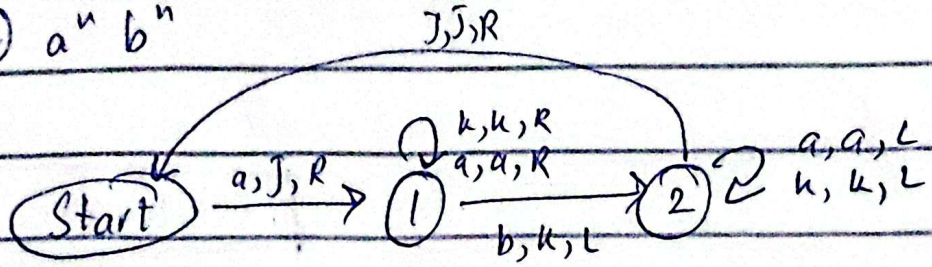
direction

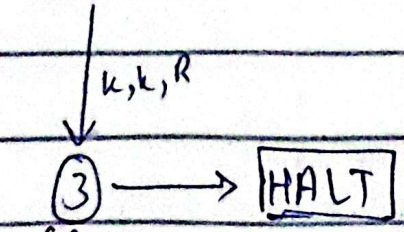
$TM = \{ \Sigma, \Gamma, Q, \delta(r, w, d), Tape \}$

① ab^*

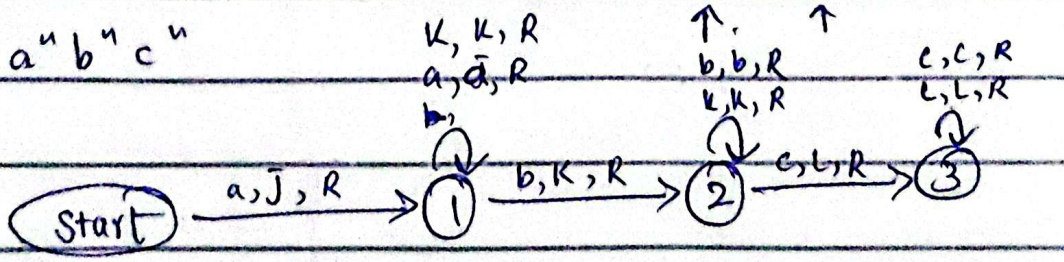


② $a^n b^n$





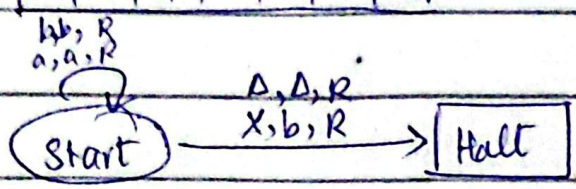
③ $a^n b^n c^n$



Turing Machine

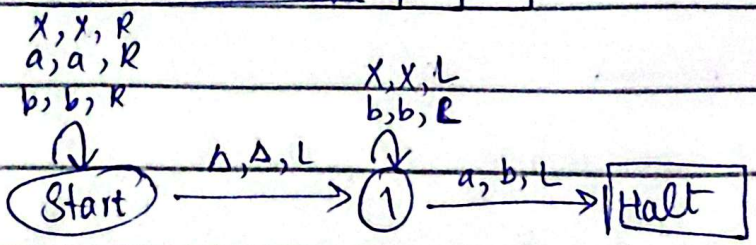
→ Replace X with b (Search X)

a	b	X	a	a	b	Δ

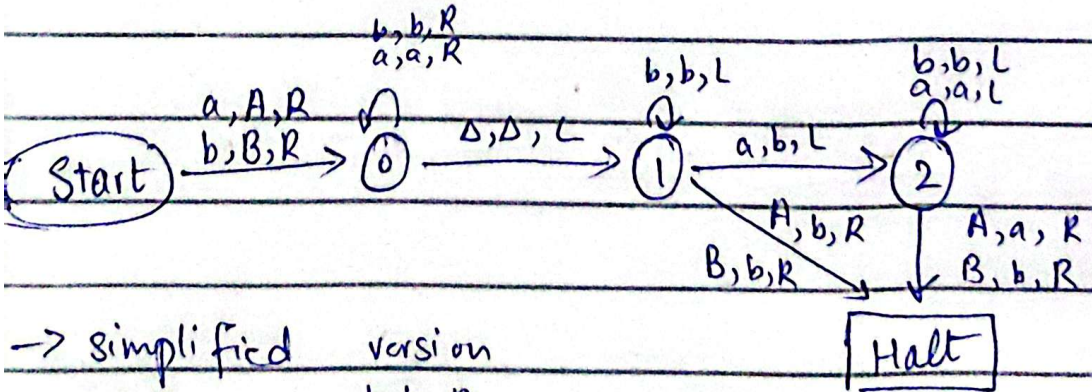


→ Replace last ~~occurance~~ occurance of a ~~replace~~ with b

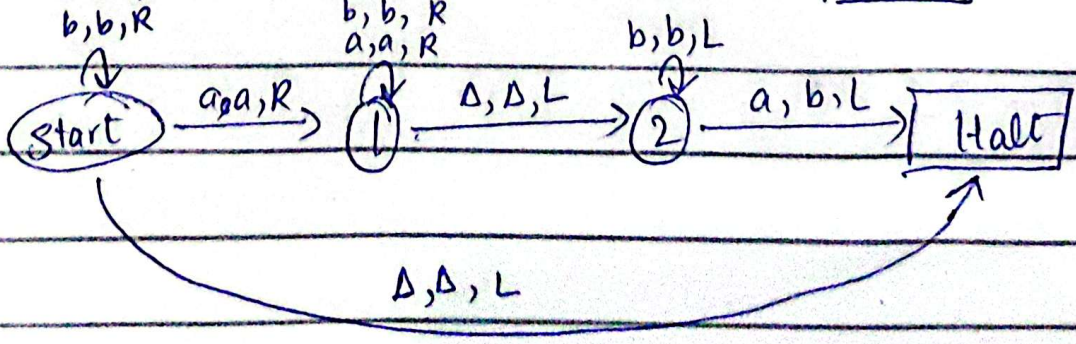
a	b	b	X	a	a	b	Δ



→ If there are no 'a' in string then

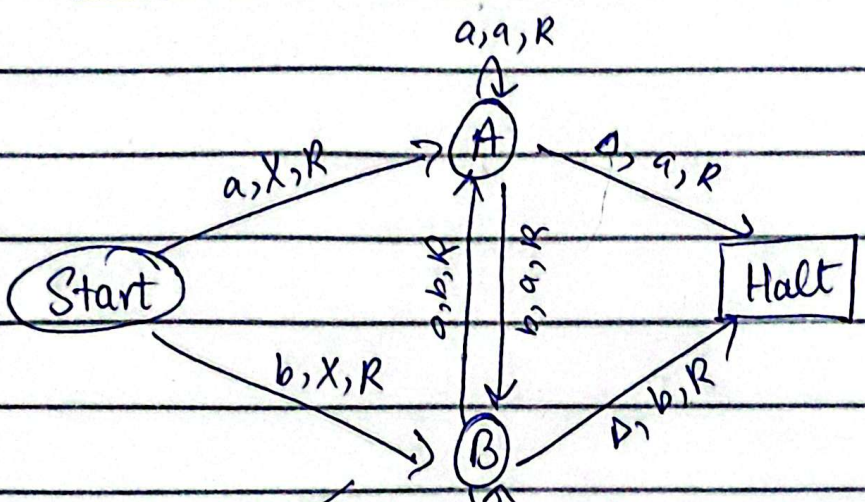


→ simplified version



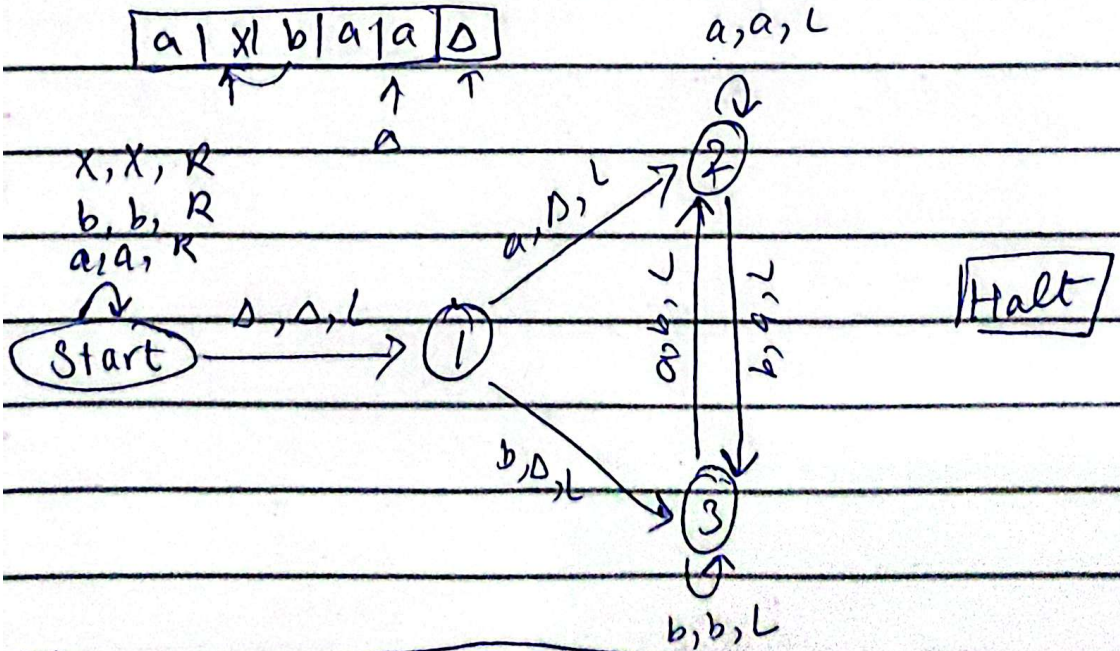
→ Insert X at third index

| a | b | b | a | a | b | Δ | → | a | b | X | b | a | a | b | Δ |

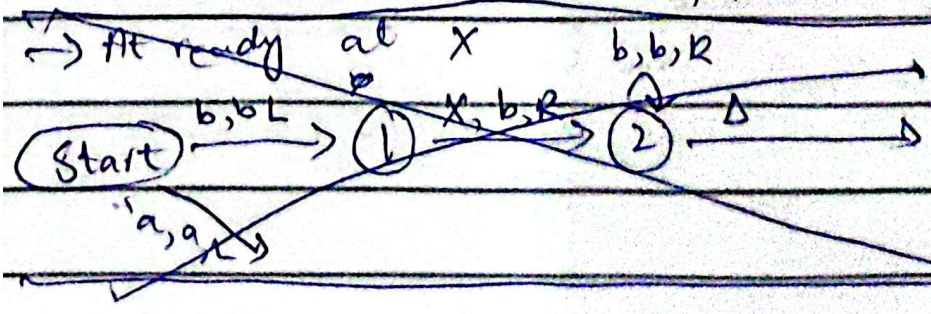


→ Delete X at

| a | X | b | a | a | Δ |



→ At ready at X

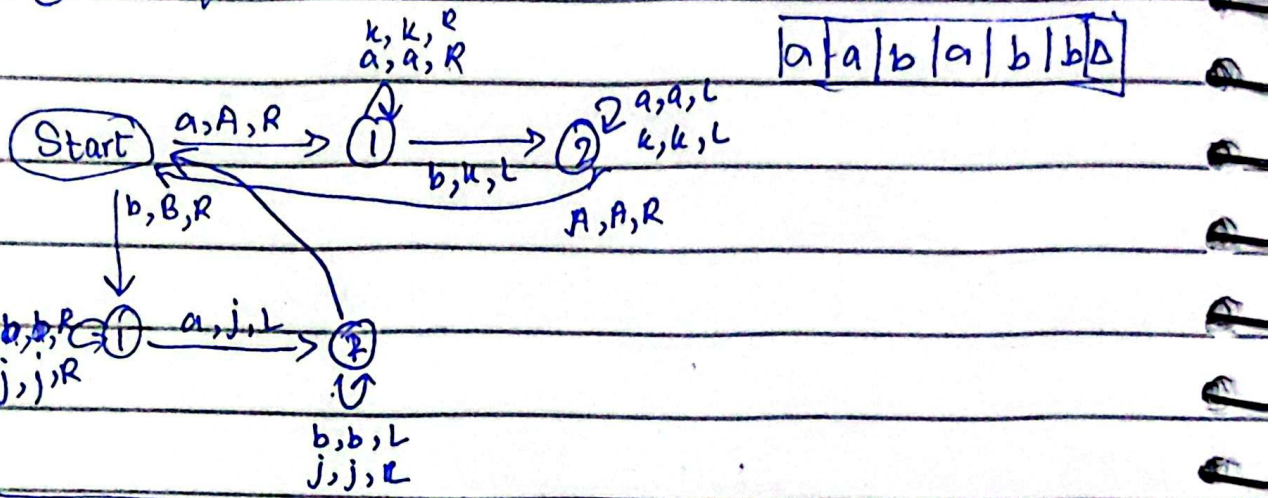


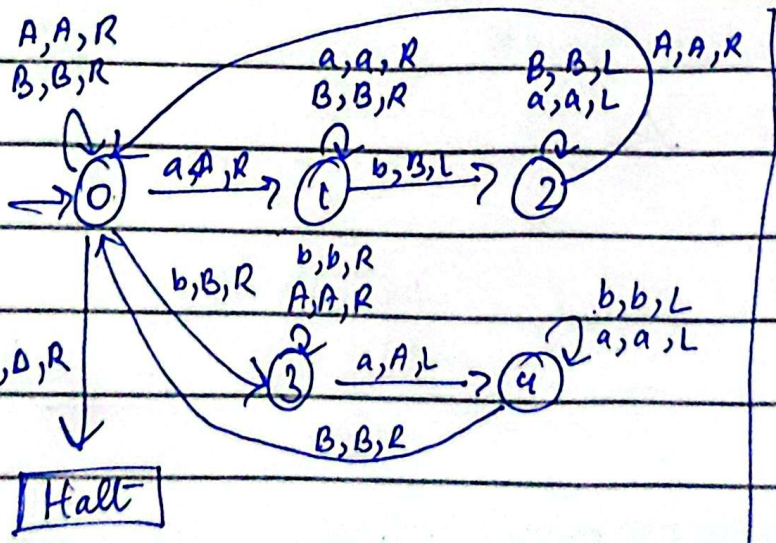
ToA

→ Read Decidability

→ Turing Machine

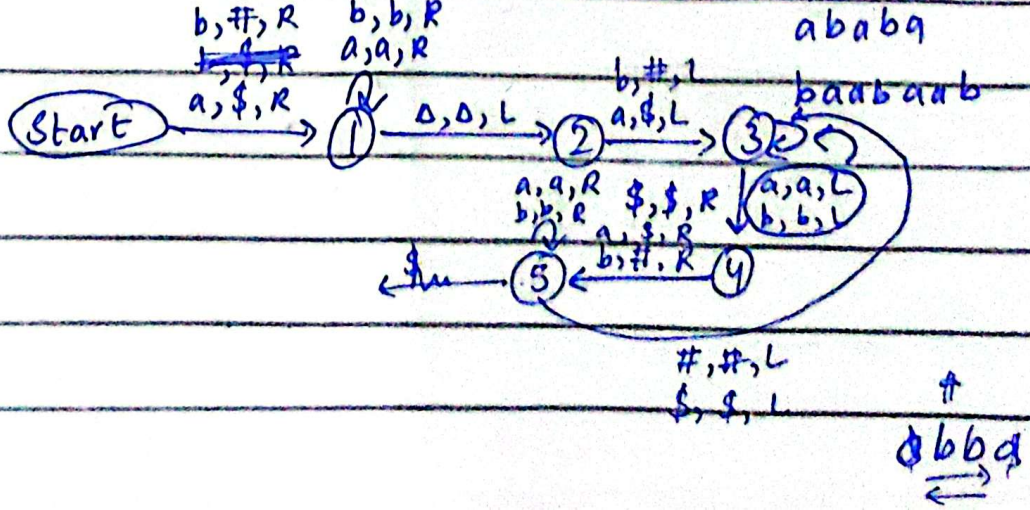
(1) → Equal a's and b's





abba
↓
B A
#

→ Palindrome



abba$
abaaba
baabaab
$bbaa