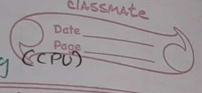
- Chapter : Sampling Methods.

9/9/24/Mon

GAYATHRI B NAIR
AM.EN.U4AIE22117

Formal Language & Automata - 22AIE302

Language
- Natural Language → Alphabet → Word. —Grammar→ Sentences.
- Formal Language (Pgmming Language) → Symbols (Σ) → String (w) —(G) Syntax→ Programmes.



Finite (FA) Automata — comp+ no memory.

Push Down (PDA) Automata — comp+ memory is a stack

Turing Machine (TM) — comp+ memory = RAM

All the above 3 are different based on the memory. The computer part remains same for all 3.

10/9/24 Tue

Membership Problem:

- Containment problem.
- Whether a set 'A' has element 'x' in it.
- set of all subsets of set A = power set of A
  if n → no: of elements in A
  then 2^n → no: of elements in P(A)

Formal Language: (L)

- They are the abstract representation of pgmming languages.
- It is basically a set of strings.

Elements in L:

1. **Alphabet: (Σ)**
   - finite, non-empty set of symbols.
   - eg: binary alphabet: $\Sigma\{0, 1\}$
     $L = \{0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$

   English alphabet: $\Sigma = \{a, b, c, \ldots, z\}$;
   $L = \{a, b, \ldots, z, ab, ac, \ldots\}$

2. **Word / String (w):**
   - It is a collection of symbols or alphabets.
   - eg: $\Sigma = \{0, 1\}$
     $w_1 = 0110$,
     $w_2 = 100$, etc.

Properties of strings:

a) **Length of a string:** - no: of characters in the string $|w|$
   - eg: $w_1 = 0110$; $|w_1| = 4$

b) **Reverse of a string:** - reverse the string w^R
   - $w_1 = 011$; $w_1^R = 110$

c) **Concatenation:** - Joining two strings $w_1 \cdot w_2$
   - $w_1 = 01$ $w_2 = 110$ $w_1 \cdot w_2 = 01110$
     $w_2 \cdot w_1 = 11001$

d) Empty/Null string: - string with no characters ε (λ)
- $|\varepsilon| = 0$
- $w\varepsilon = \varepsilon w = w$

12/9/24 Thursday.

e) Substring: - w = abcde
- substrings: a, b, c, d, e
ab, bc, cd, de,
abc, bcd, cde
abcd, bcde
abcde
- any portion of a string.

f) Prefix & suffix: - $w = uv$
prefix ↵ ↳ suffix
- eg: w = abcde

	prefix	suffix
proper prefix / proper suffix	a, ab, abc, abcd,	e, de, cde, bcde,
	abcde	abcde
	ε	ε

We can add epsilon (empty string) anywhere so it comes as both prefix and suffix
All prefixes or suffixes except the original string is called proper prefix or suffix respectively. This includes ε.

Power of an alphabet: (Σ^k)

eg: binary alphabet: $\Sigma = \{0, 1\}$

$\Sigma^2 = \{00, 01, 10, 11\}$
$\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

The power here stands for the length of the string.

ie; if length of string = k, then, Σ^k is the set of all such strings with length k.

★ $\Sigma^0 = \{\varepsilon\}$
$\Sigma^1 = \{0, 1\}$

Star Closure (Σ^*)

- Set of all strings from alphabet Σ including ε.
- $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$
↳ union

Σ^+

- Set of all strings except epsilon (ε)
- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$

Complement of a Language:

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

$\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, \ldots\}$

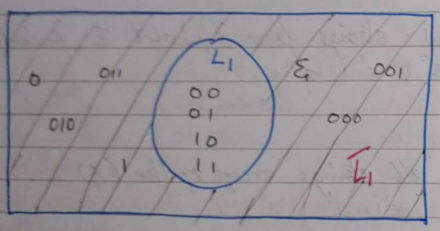
Let L_1 be a language with alphabet having 2 chars only. Then,

$L_1 = \{00, 01, 10, 11\}$

Now $\bar{L}_1$ (L_1 complement) is:

$\bar{L}_1 = \{\varepsilon, 0, 1, 000, 001, 010, 011, 100, \ldots\}$

$\bar{L}_1 = \Sigma^* - L_1$



Types of Automata:

	Automata	Architecture	Memory
1.	Finite Automata (FA)	computer	No memory → Only solves Membership problems
2.	Push Down Automata (PDA)	comp ↔ stack	Stack → Only solves Membership problems
3.	Turing Machine (TM)	comp ↔ RAM	RAM → solves Membership problems of some computations. (addition, substraction, mult etc)

What do membership problems mean?

A machine ~~is~~ represented ~~by~~ a language.

input: w → [FA for language L] → output: $w \in L$ / Accept; $w \notin L$ / Reject.

The language / machine checks whether the string belongs to the language and thereby chooses whether or not to accept the string.

18/9/24/ Wed.

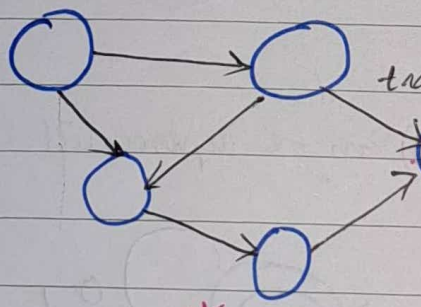
Formal Language & Automata

1. Finite Automata: (FA)

- Every finite automata represents a language.

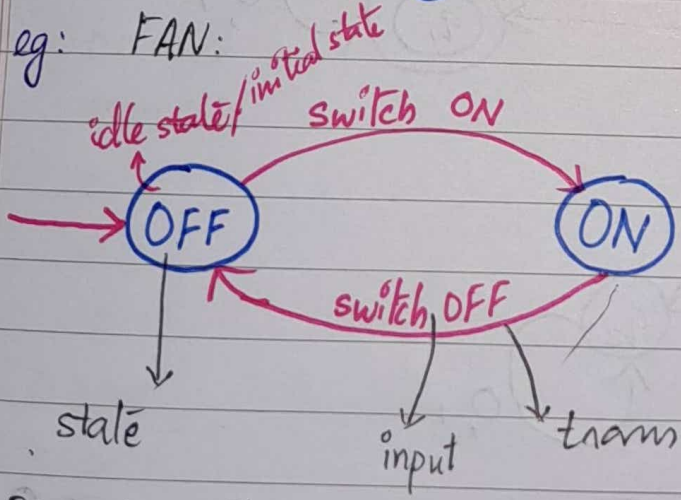
Q. How to represent or design FA?

↳ • We use directed graph to represent FA.
- This representation is called transition diagram.
- eg:

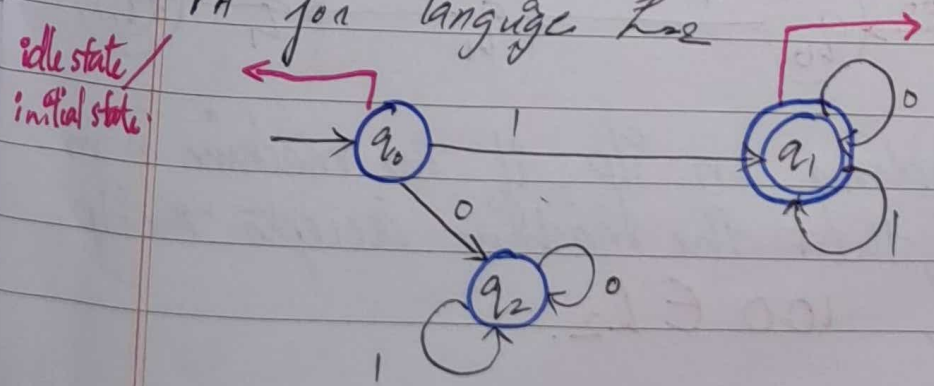


nodes: states
edges: ~~arrow~~ transitions.

- eg: FAN:



FA for language L_{eg}



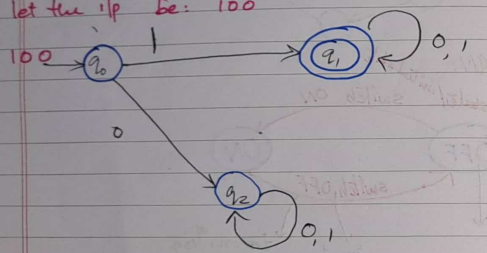
FABER-CASTELL
Date
Page No.

Types of states

1. Initial state
   - single & unique. → q_0
2. Final state
   ~~- at least~~ q_n
   - one or more
3. other states
   - intermediate state q_n
   - 0 or more

The language $FA(L_2)$ can also be represented as

let the i/p be: 100

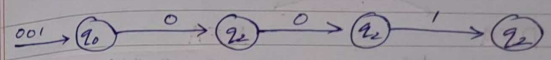


~~start~~ state: 100 → q_0 —1→ q_1 —0→ q_1 —0→ q_1

If after reading an i/p, if the machine is in final state, then the machine accepts the i/p.
That means, $100 \in L_2$.

FABER-CASTELL
Date
Page No.

i/p = 001



↳ not final state

So $001 \notin L_2$.

i/p	Accept / Reject
0	R
1	A
00	R
01	R
10	A
11	A
000	R
~~00~~ 101	~~R~~ A
010	R
011	R
100	A
0000	R
0101	R
1010	A
1100	A
1001	A

$L_2 = \{ w \mid w$ starts with 1, $\Sigma = \{0,1\}\}$

FA

DFA
Deterministic FA
exactly one transition for every i/p symbol.

NFA
Nondeterministic FA
zero or more transitions for every i/p symbol

eg:

formal definition

DFA:

It is defined as a 5 tuple

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

Q - finite set of states
eg: $Q = \{q_0, q_1, q_2, q_3\}$

Σ - finite set of i/p symbols / alphabet
eg: $\Sigma = \{0, 1\}$

δ - transition function which maps
Current state $\leftarrow Q \times \Sigma \rightarrow Q$ next state
current i/p

q_0 - initial state

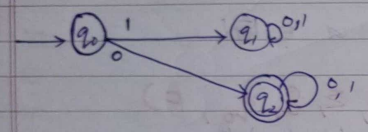
F - finite set of final states.
$F \subseteq Q$

eg.. $\delta(q_0, 1) = q_1$
$\delta(q_1, 1) = q_1$
$\delta(q_1, 0) = q_2$ (for the Σ_2 FA)

23/09/24/Mon.

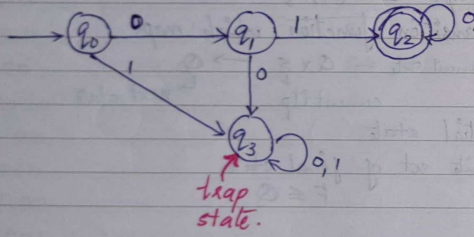
Q. Design a DFA that accepts all strings starting with zero, $\Sigma = \{0, 1\}$?

A:



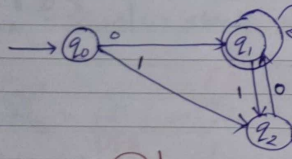
Q. Design a DFA that accepts all strings starting with '01', $\Sigma = \{0, 1\}$.

A:

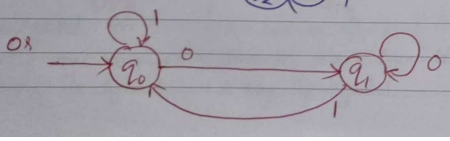


Q. What about those ending in zero?

A:

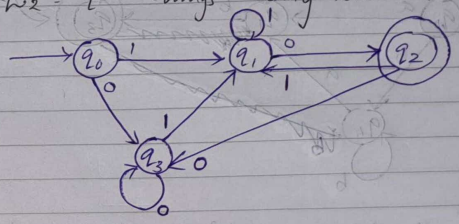


or



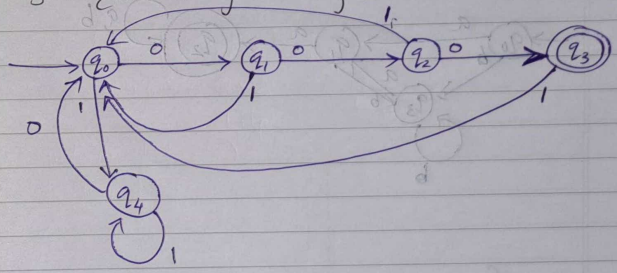
Q. $L_2 = \{$ all strings ending with '10' $\}$

A:



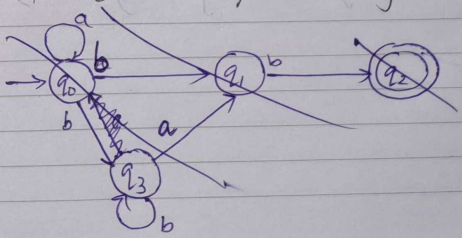
Q. $L_3 = \{$ all strings ending with '000' $\}$

A:

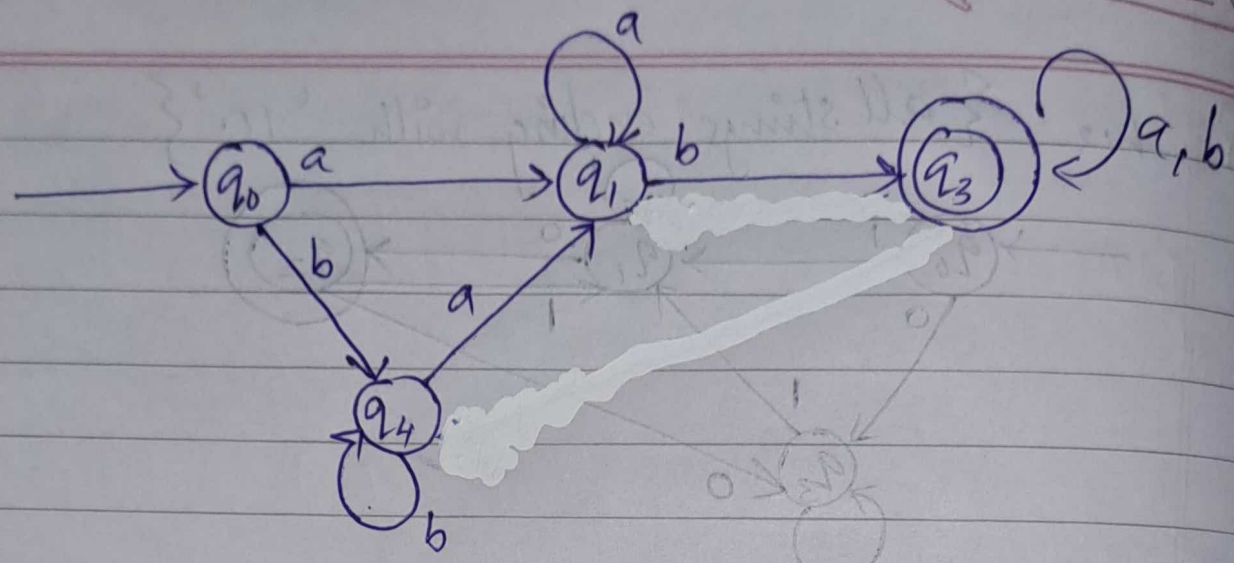


Q. $L_4 = \{$ all strings containing 'ab', $\Sigma = \{a, b\}\}$

A:

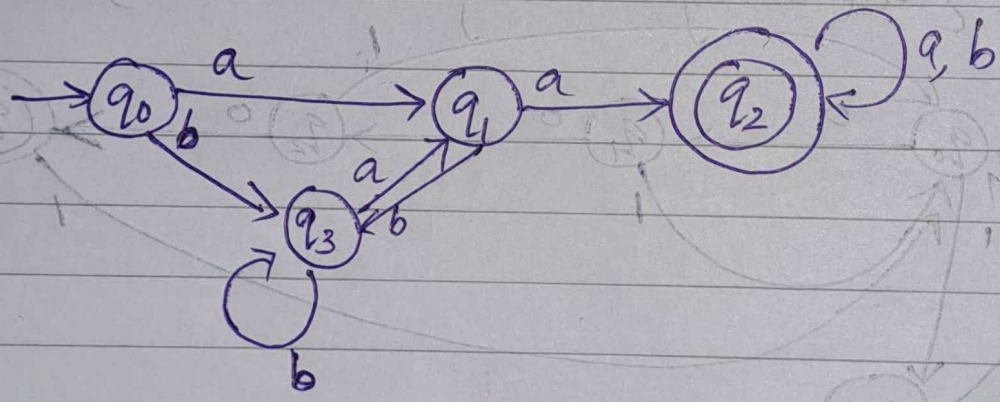


A:



Q. $L_5 = \{$all strings containing atleast 2 a's$\}$.

A:



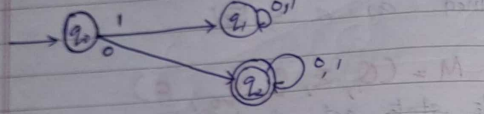
Q. $L_6 = \{$ all strings containing atmost 2 a's$\}$

Q. $L_7 = \{$ all strings containing exactly 2 a's$\}$

23/09/24/Mon.

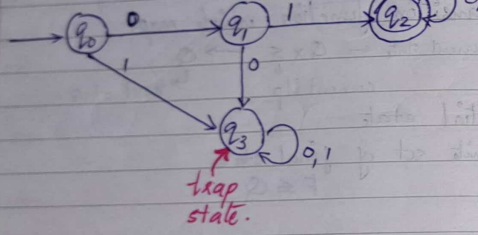
Q. Design a DFA that accepts all strings starting with zero, $\Sigma = \{0, 1\}$?

A:



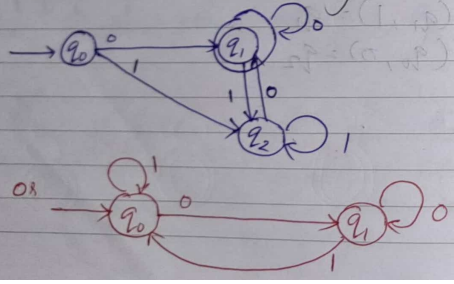
Q. Design a DFA that accepts all strings starting with '01', $\Sigma = \{0, 1\}$.

A:



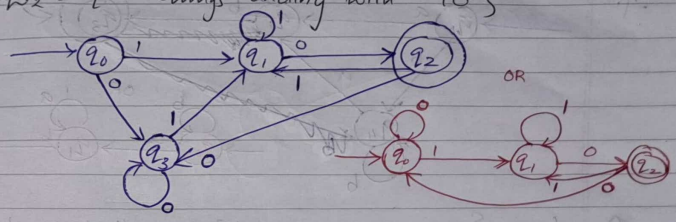
Q. What about those ending in zero?

A:



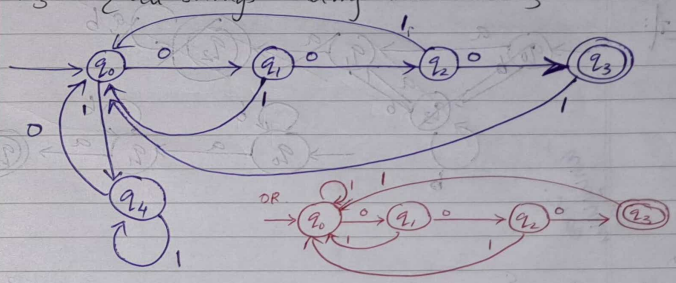
Q. $L_2 = \{$ all strings ending with '10' $\}$

A:



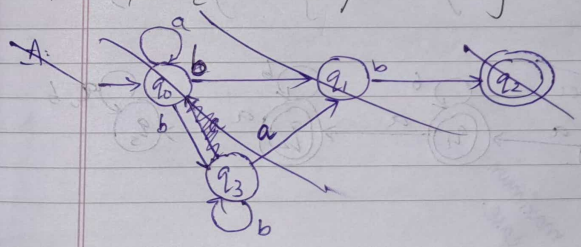
Q. $L_3 = \{$ all strings ending with '000' $\}$

A:

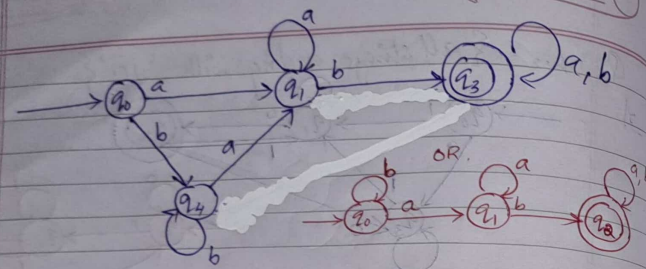


Q. $L_4 = \{$ all strings containing 'ab', $\Sigma = \{a, b\}\}$

A:

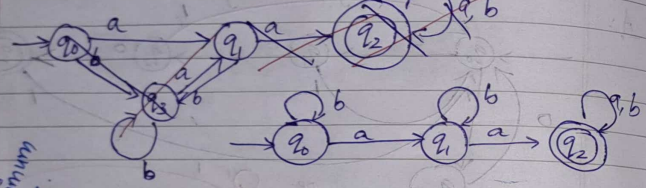


A:



Q. L_5 = {all strings containing atleast 2 a's}.

A:



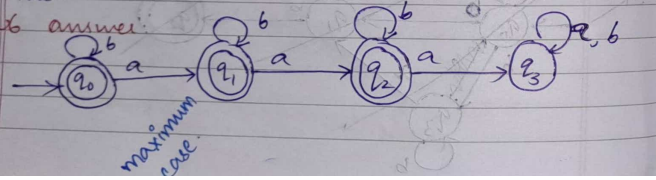
24/8

Q. L_6 = { all strings containing atmost 2 a's}

Q. L_7 = { all strings containing exactly 2 a's}

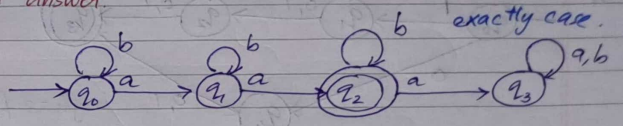
24/08/24 Tue

L_6 answer:



L_7 answer.

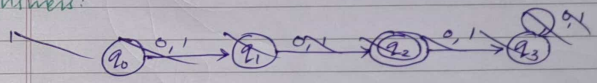
exactly case.



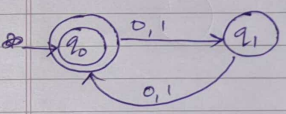
Q. Design DFA for the following:

1. L_1 = { all strings of even length } $\Sigma = \{0,1\}$
2. L_2 = { all strings with length == 3 } $\Sigma = \{0,1\}$
3. L_3 = { all strings with length: the multiple of 3 } $\Sigma = \{0,1\}$
4. L_4 = {all strings with length 3 and starts with 1 } $\Sigma = \{0,1\}$

Answers:

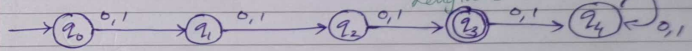


1.



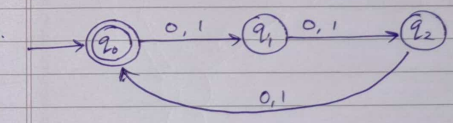
multiple of 3

2.



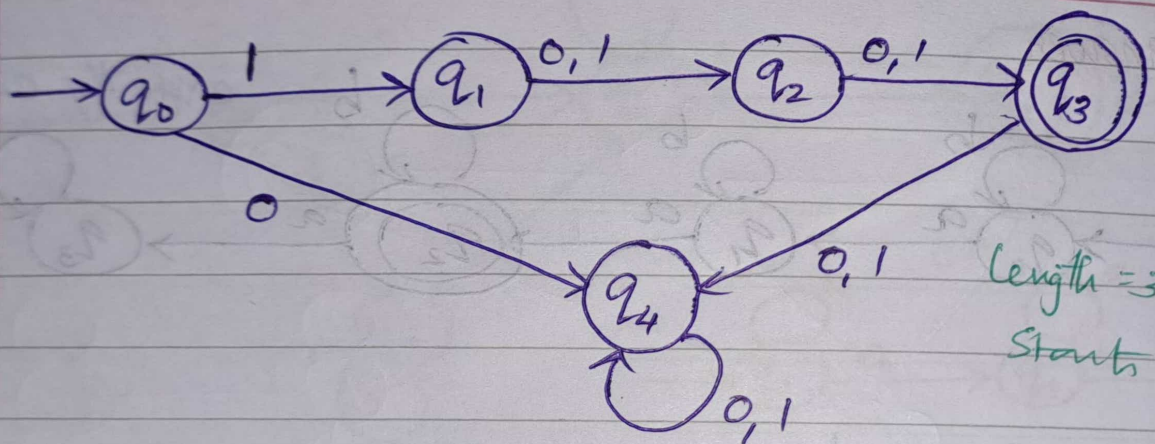
Length = 3

3.



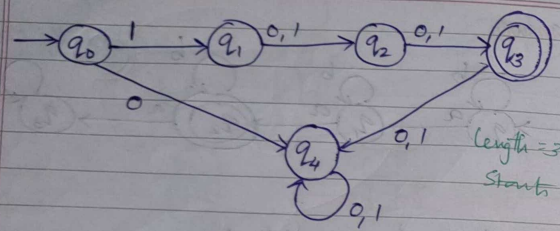
Multiple of 3

4.



Length = 3 &
Starts with 1

4.



Length = 3 & Starts with 1

03/10/24 | Thursday.

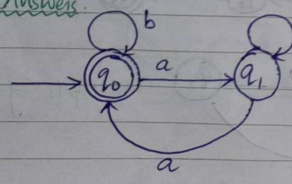
Tutorial - 1 :

Design DFA for the following languages

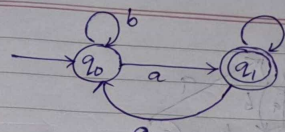
1. L_1 = {all strings with even no: of a's}, $\Sigma = \{a, b\}$
2. L_2 = {all strings with odd no: of a's}, $\Sigma = \{a, b\}$
3. L_3 = {all strings with even no: of a's & odd no: of b's}, $\Sigma = \{a, b\}$
4. ~~L_4 = {all strings with}~~
4. L_4 = {all binary strings divisible by 4}, $\Sigma = \{0, 1\}$
5. L_5 = {all decimal no:s divisible by 5}, $\Sigma = \{0, 1, 2 \ldots, 9\}$
6. L_6 = { $n_a(w) \% 3 = 0$ and $n_b(w) \% 2 = 0$ }, $\Sigma = \{a, b\}$
7. L_7 = {all binary strings not divisible by 4}, $\Sigma = \{0, 1\}$

Answers.

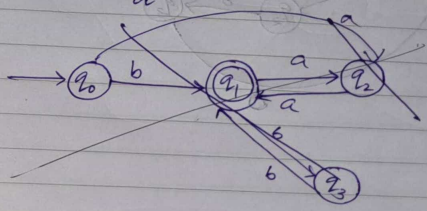
1.



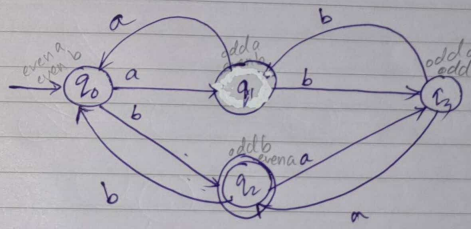
2.



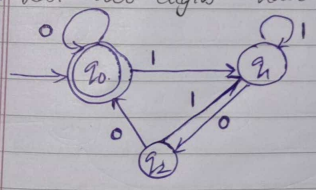
3.



4.



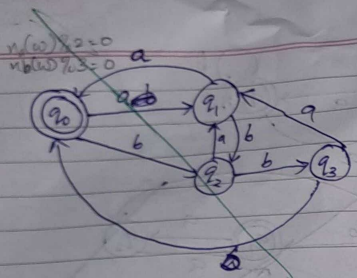
4. Last two digits have to be zero.



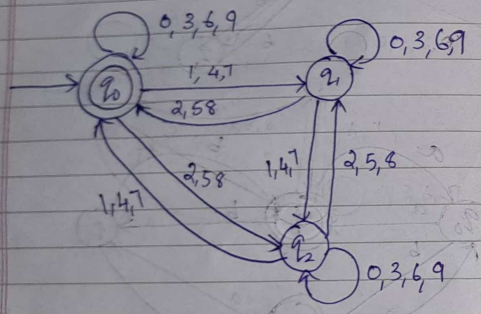
0	0
1	1
10	2
11	3
100	0
101	1
110	2
111	3
1000	0
1100	0

10101100

6.



5.

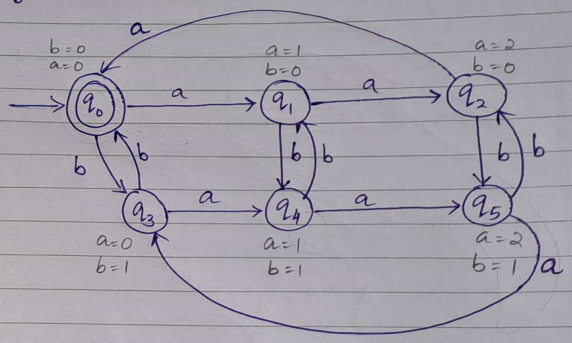


1, 4, 7,
1, 4, 7
2, 8, 11
1, 4, 7
3, 12, 21

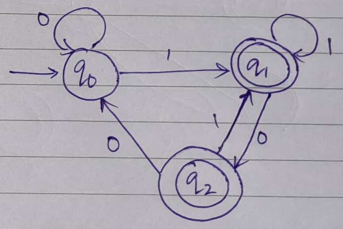
3 groups: {0, 3, 6, 9;
1, 4, 7
2, 5, 8

6.

6. $n_a(w)\%3=0$ States: a=0 b=0 ; a=1 b=0 ; a=2 b=0
$n_b(w)\%2=0$ a=0 b=1 ; a=1 b=1 ; a=2 b=1



7.



7/10/24 Monday.

Non-deterministic finite Automata:

Definition: NFA is defined as a 5-tuple

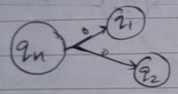
$M = (Q, \Sigma, \delta, q_0, F)$

Q: ~~finite~~ finite set of states

Σ: finite set of alphabet / i/p symbol

δ: it maps $Q \times \Sigma \longrightarrow 2^Q$ (set of subsets)

This means there can be multiple transitions

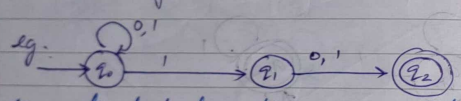


$\delta(q_n, 0) = \{q_1, q_2\}$

q_0: initial state

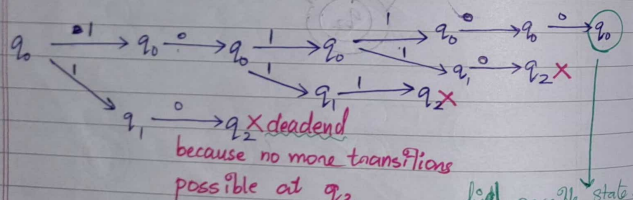
F: final state.

eg:



How to check if a string is accepted in NFA?

$w = 101100$

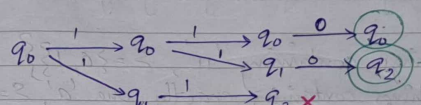


because no more transitions possible at q_2

final possible state.

This is not final state and so this word ~~does~~ is not accepted ~~belong to this diff~~

- $w = 110$.

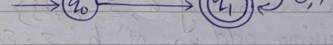


This is a final state so it is accepted.

Design NFA for the following languages

1. $L_1 = \{$ all strings starts with 0, $\Sigma = \{0, 1\}\}$

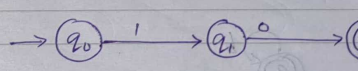
A:



Here we need not bother with transitions that we do not prefer (such as what to do if 1 came as first digit)

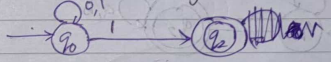
2. $L_2 = \{$ all strings starts with '10', $\Sigma = \{0, 1\}\}$

A:



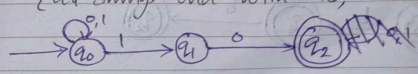
3. $L_3 = \{$ all strings end with 1, $\Sigma = \{0, 1\}\}$

A:



4. $L_4 = \{$ all strings end with '10', $\Sigma = \{0, 1\}\}$

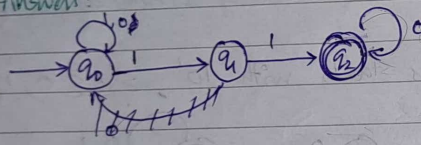
A:

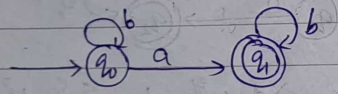


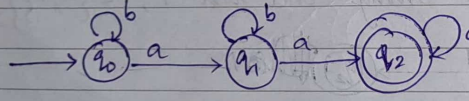
Design NFA for the following languages:

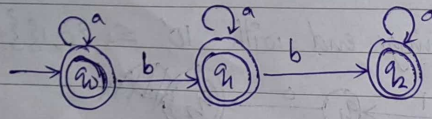
1. L_1 = {all strings containing '11', $\Sigma = \{0, 1\}$}
2. L_2 = {all strings containing exactly one 'a', $\Sigma = \{a, b\}$}
3. L_3 = {all strings containing atleast 2 'a', $\Sigma = \{a, b\}$}
4. L_4 = {all strings containing atmost two 'b', $\Sigma = \{a, b\}$}
5. L_5 = {all strings with length multiple of three, $\Sigma = \{0, 1\}$}
6. L_6 = {all strings with even length, $\Sigma = \{0, 1\}$}
7. L_7 = {all strings with even no: of 'a', $\Sigma = \{a, b\}$}
8. L_8 = {all strings containing odd no: of 'b', $\Sigma = \{a, b\}$}

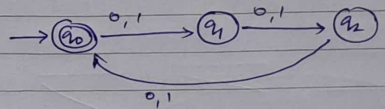
Answers:

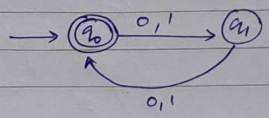
1.


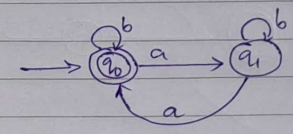
2.


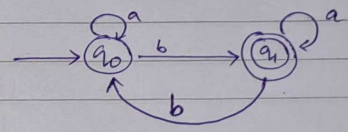
3.


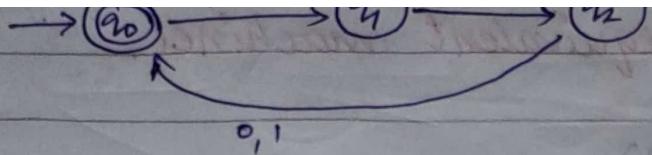
4.


5.


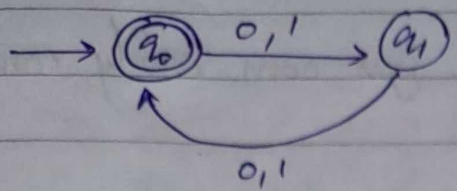
6.


7.


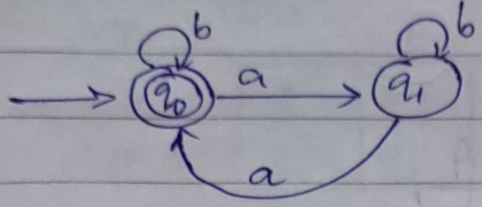
8.




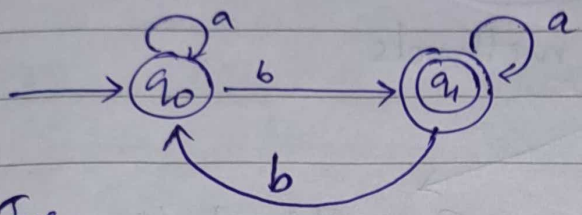
6.



7.



8.

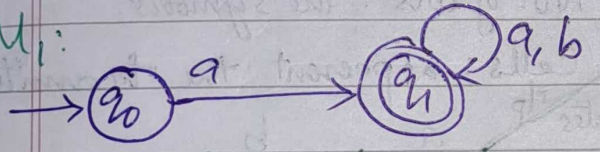


8/10/24/Tue.

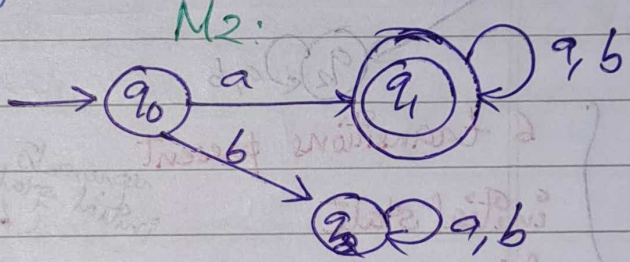
Equivalence of NFA & DFA:

* DFA is a special case of NFA.

M_1:



M_2:



M_1 : NFA M_2 : DFA

both M_1 & M_2 accepts all strings starting with 'a'.

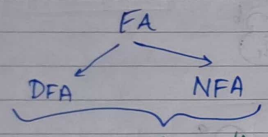
When two machines accept the same language, they are called equivalent machines.

ie, $M_1 \equiv M_2$

(NFA) (DFA)

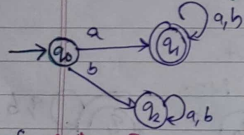
or in other words,

if you have an NFA, you can easily convert it to a DFA.



representation methods

Transition diagram



6 transitions present
- initial state
- final state
- the other states
- i/p symbols

(i/p: we get from the diagram)

Transition table

↓

No: of rows: the no: of states present

No: of cols: the symbols.

Cells: represent the transition.

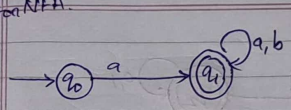
states \ i/p	a	b
→ q_0	q_1	q_2
* q_1	q_1	q_1
q_2	q_2	q_2

(→ represents initial state; * represents final state)

→ : initial state

* : final state.

Start NFA.



states \ input	a	b
→ q_0	q_1	–
* q_1	q_1	q_1

or,

	$\{q_1\}$	$\{\}$
	$\{q_1\}$	$\{q_1\}$

we can also write in set representation

Q. How to convert NFA to DFA?

↳ Step 1: Draw the transition table for NFA.

states \ input	a	b
→ q_0	q_1	$\{\}$
* q_1	q_1	q_1

Step 2: Write the transition table for DFA required.

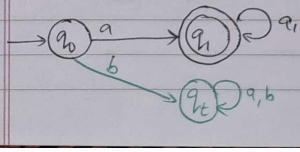
If blank: introduce a trap state

AND always start from initial state.

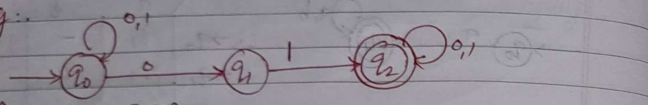
states \ input	a	b
→ q_0	q_1	q_2
* q_1	q_1	q_1
q_2	q_2	q_2

Add the trap state onto the transition & make itself the transition for every i/p

⇓ Machine form (Diagram)



eg:



Convert to DFA:

A: Step 1:

	0	1
→ q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{\}$	$\{q_2\}$
* q_2	$\{q_2\}$	$\{q_2\}$

Step 2:

	0	1
→ q_0	q_{01}	q_0
q_{01}	q_{01}	q_{02}
* q_2	/ / /	/ / /

There are two transitions. So we write a new state combining the both. Now we go to the new state

	0	1
q_0	q_0, q_1	q_0
q_1	–	q_2
	$\{q_0, q_1\}$	$\{q_0, q_2\}$
	↓	
	q_{01}	q_{02}

Take union

Now define q_{02}

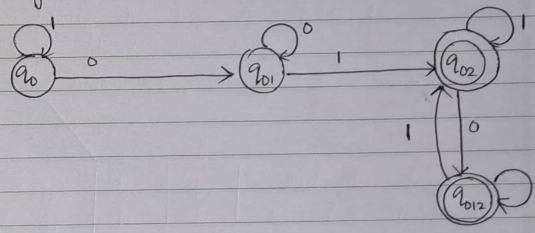
	0	1
→ q_0	q_{01}	q_0
q_{01}	q_{01}	q_{02}
* q_{02}	q_{012}	q_{02}
* q_{012}	q_{012}	q_{02}

All states have been defined.

Now which is the final state?

Originally q_2 is the final state. So all states with q_2 is the final state.

The diagram will be:



	0	1
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_2	$\{q_2\}$	$\{q_2\}$
	$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$
	q_{012}	q_{02}

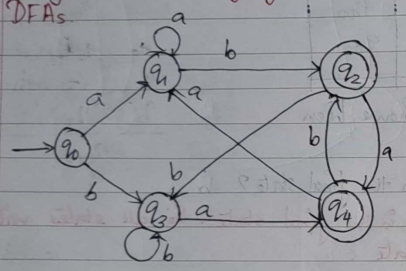
q_0	q_0, q_1	q_0
q_1	–	q_2
q_2	q_2	q_2
	q_{012}	q_{02}

10/10/24 | Thu.

Tutorial-2 (DFA)

1. Recognise the languages accepted by the following DFAs.

a)

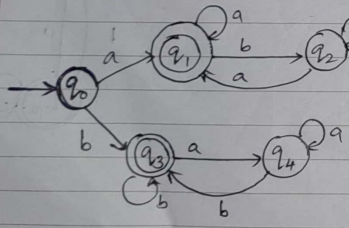


A: {All strings ending with "ab" or "ba"}

or.

$L_1 = \{w \mid w$ ends with "ab" or "ba", $\Sigma = \{a,b\}\}$

b)

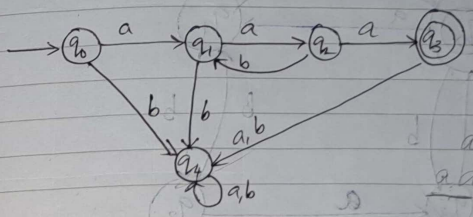


A:

$L_2 = \{w \mid w$ ends with the starting letter, $\Sigma = \{a,b\}\}$

aababaaa

c)



- aaa together
- starts with aa ends with exactly one baa
- no 2 bs come together.

So $aa(ba)^n a$, $n \geq 0$

(aaa) : $aa(ba)^n a$

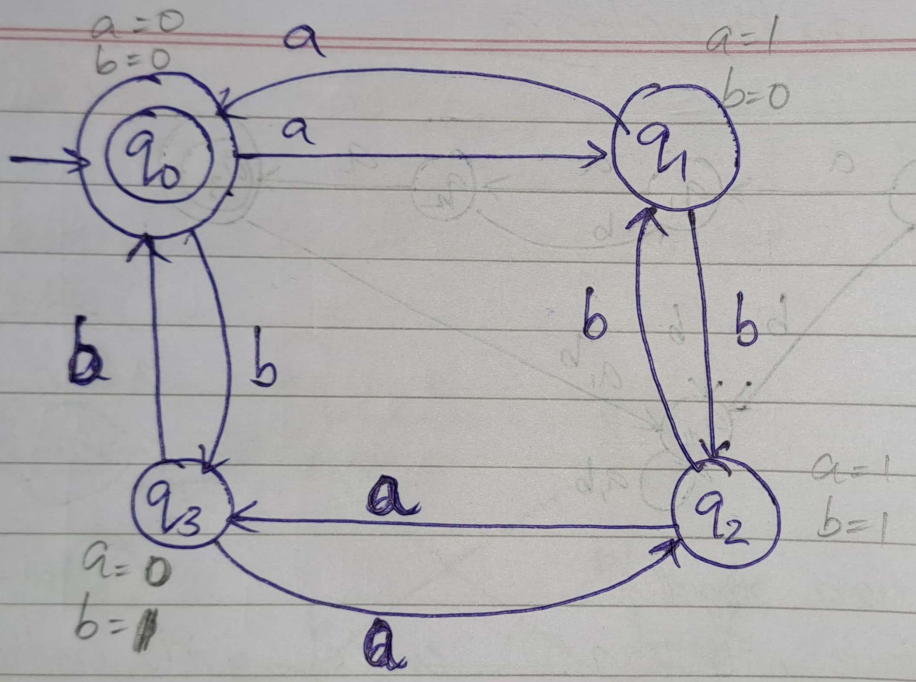
A: ~~$L_3 = \{w \mid w$ is either 'aaa' or of the form $aa(ba)^n a$, $n \geq 0$ $n \in W$; $\Sigma = \{a,b\}\}$~~

A: $L_3 = \{w \mid w$ is of the form $aa(ba)^n a$, $n \geq 0$ $n \in W$; $\Sigma = \{a,b\}\}$

II. Design DFA to recognise strings in the following languages.

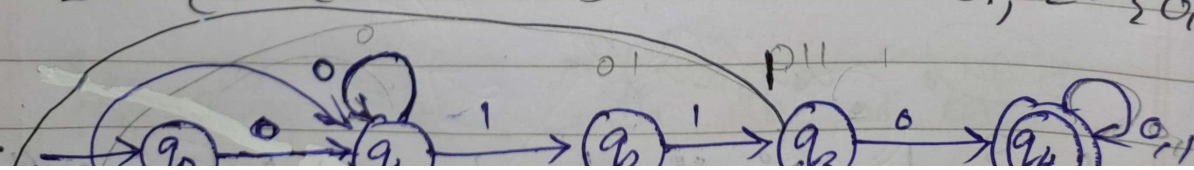
a) $L_1 = \{w \mid w$ contains even no: of a's and even no: of b's, $\Sigma = (a,b)\}$

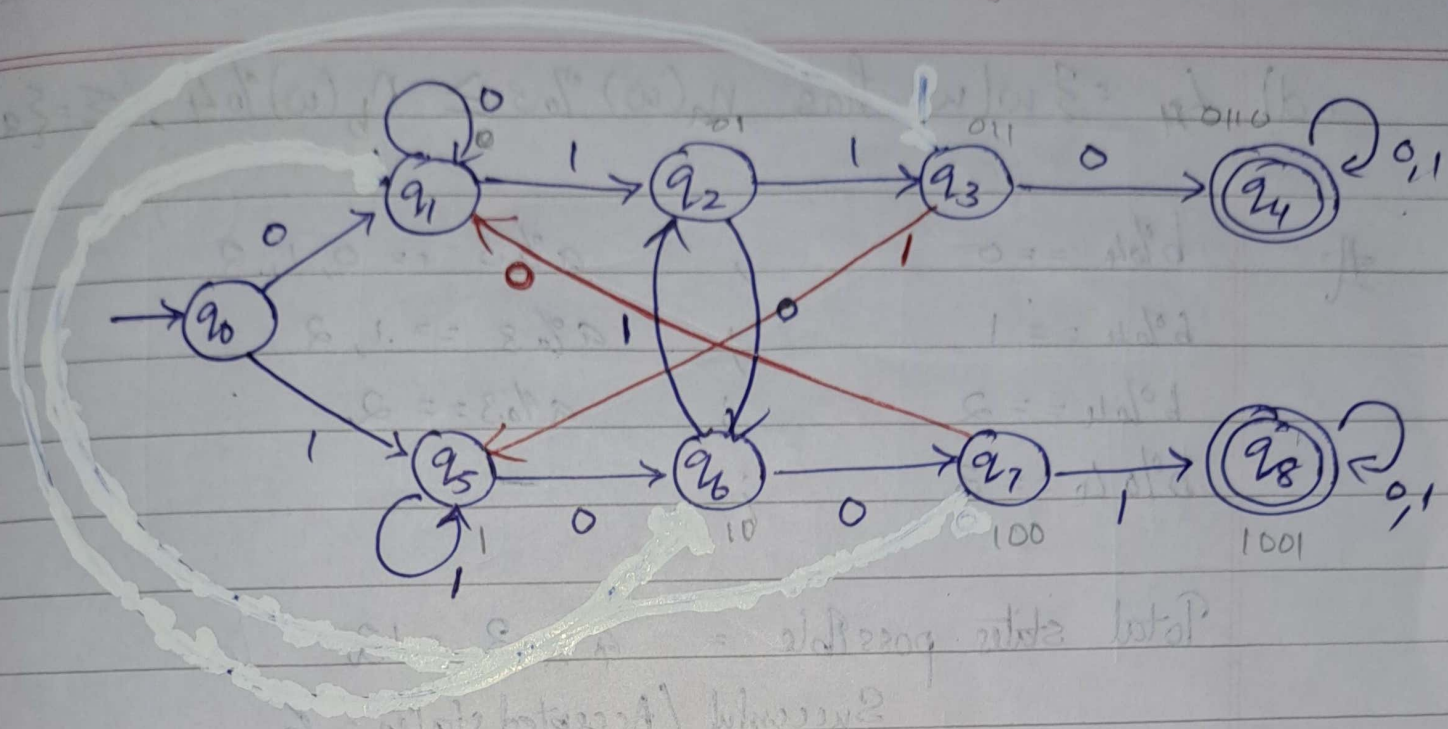
A:



b) $L_2 = \{ w \mid w \text{ contains } 0110 \text{ or } 1001, \Sigma = \{0, 1\} \}$

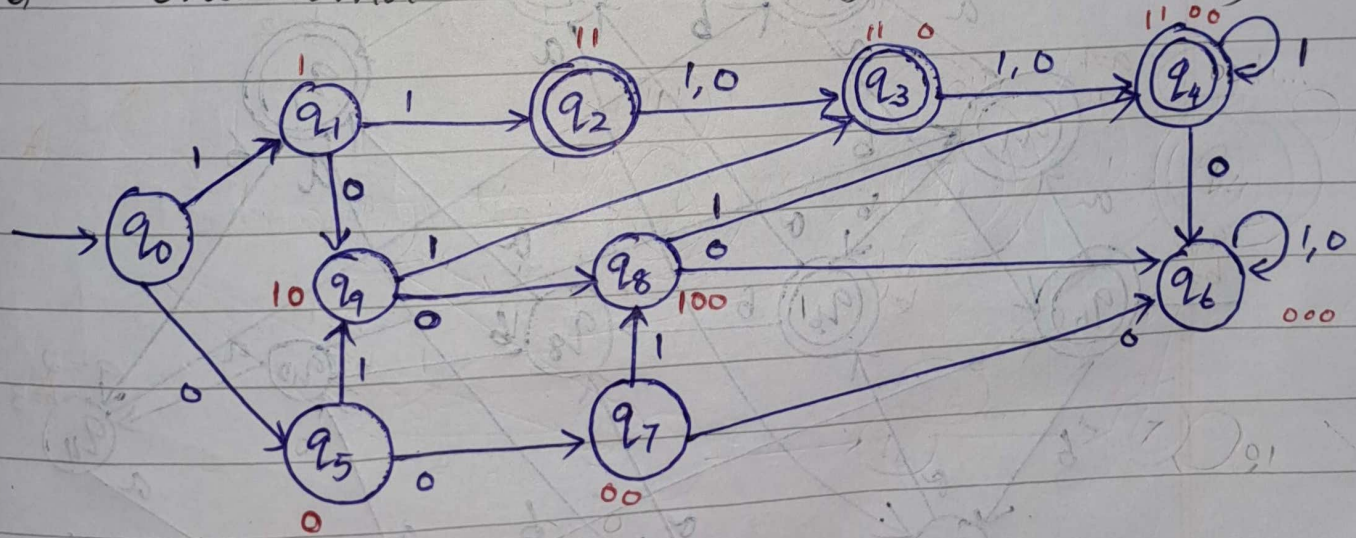
A:





c) $L_3 = \{w \mid w$ is a binary string with atleast two ones and almost two zeroes, $\Sigma = \{0,1\}\}$

A:



Possible states : initial,

1	100 . 000 → trap.
0	00
10	110
11	1100

d) $L_4 = \{w \mid w \text{ has } n_a(w) \%3 \geq n_b(w) \%4, \Sigma = \{a,b\}\}$

A:

$b\%4 == 0$; $a\%3 == 0, 1, 2$

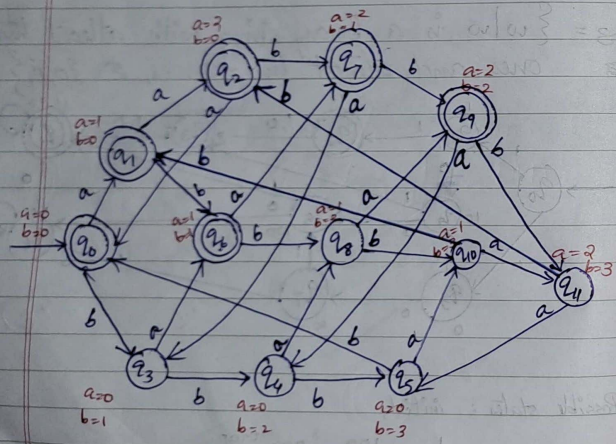
$b\%4 == 1$; $a\%3 == 1, 2$

$b\%4 == 2$; $a\%3 == 2$

$b\%4 == 3$; –

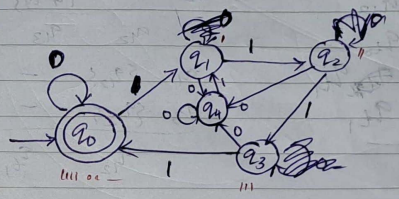
Total states possible = $4 \times 3 = 12$ etc

Successful / Accepted states = 6



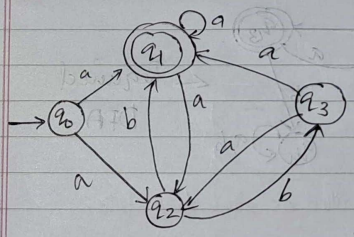
e) $L_5 = \{w \mid$ No. of consecutive 1's in w is 0 or multiple of 4, $\Sigma = \{0,1\}\}$.

A:



III. Convert the following NFA to DFA using Subset Construction method.

a)



A:

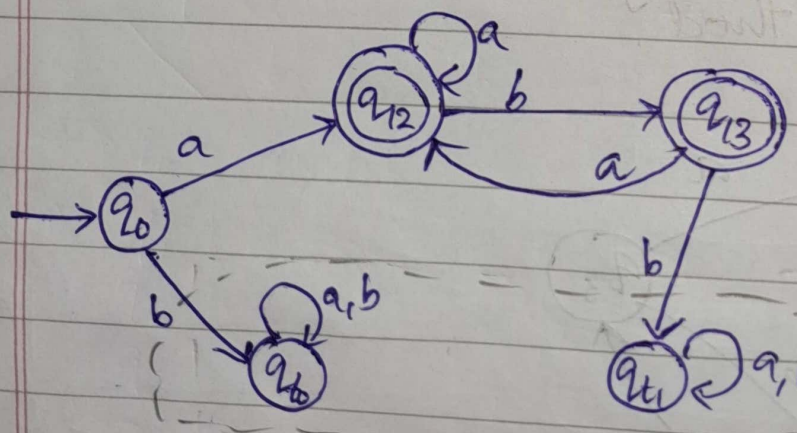
	a	b
→ q_0	$\{q_1, q_2\}$	–
* q_1	$\{q_1, q_2\}$	–
q_2	–	$\{q_1, q_3\}$
q_3	$\{q_1, q_2\}$	–

→ transition table of NFA given.

	a	b
→ q_0	q_{12}	q_{t_0}
* q_{12}	q_{12}	q_{13}
q_{t_0}	q_{t_0}	q_{t_0}
* q_{13}	q_{12}	q_{t_1}
q_{t_1}	q_{t_1}	q_{t_1}

q_1	$\{q_1 q_2\}$	–
q_2	–	$\{q_1 q_3\}$
–	q_{12}	q_{13}
q_1	$\{q_{12}\}$	–
q_3	$\{q_{12}\}$	–
	q_{12}	q_{t_1}

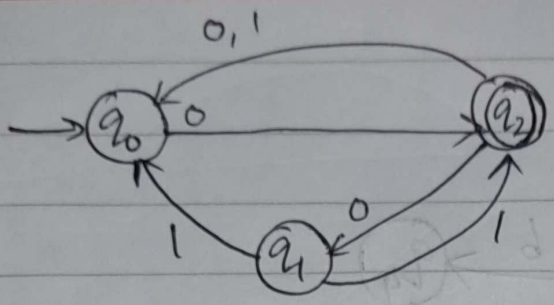
↓ diagram.



← Required DFA.

Can be written as single trap state.

b)



A: Transition table of NFA:

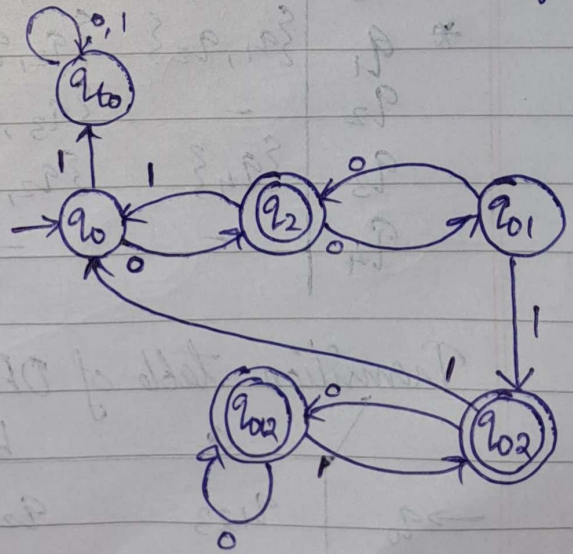
	0	1
→ q_0	$\{q_2\}$	–
q_1	–	$\{q_0, q_2\}$
* q_2	$\{q_0, q_1\}$	$\{q_0\}$

DFA transition table:

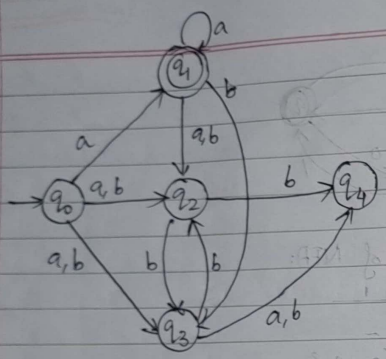
	0	1
→ q_0	q_2	q_{10}
q_{10}	q_{10}	q_{10}
* q_2	q_{01}	q_0
q_{01}	q_2	q_{02}
* q_{02}	q_{012}	q_0
* q_{012}	q_{012}	q_{02}

⇒

DFA transition diagram.



c)



A: Transition table of NFA:

	a	b
→ q_0	$\{q_0,q_1,q_3\}$	$\{q_2,q_3\}$
* q_1	$\{q_1,q_2\}$	$\{q_2,q_3\}$
q_2	–	$\{q_3,q_4\}$
q_3	$\{q_4\}$	$\{q_2,q_4\}$
q_4	–	–

Transition table of DFA:

	a	b
→ q_0	q_{123}	q_{23}
* q_{123}	q_{124}	q_{234}
q_{23}	q_4	q_{234}
* q_{124}	q_{12}	q_{234}
q_{234}	q_4	q_{234}
q_4	q_{to}	q_{to}
* q_{12}	q_{12}	q_{234}

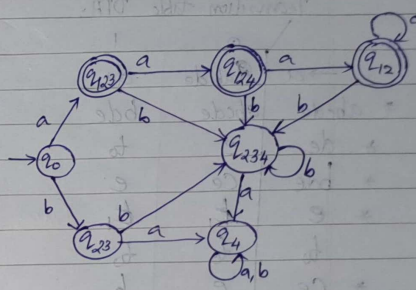
	a	b
q_{to}	q_{to}	q_{to}

Table:

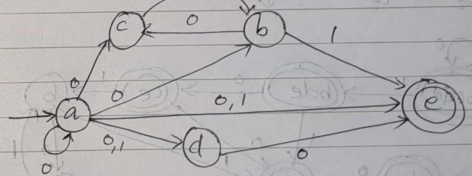
	a	b
→ q_0	q_{123}	q_{23}
* q_{123}	q_{124}	q_{234}
q_{23}	q_4	q_{234}
* q_{124}	q_{12}	q_{234}
q_{234}	q_4	q_{234}
q_4	q_{to}	q_{to}
* q_{12}	q_{12}	q_{234}
q_{to}	q_{to}	q_{to}

written as single trap state

⇒



d)



A: Transition table for NFA:

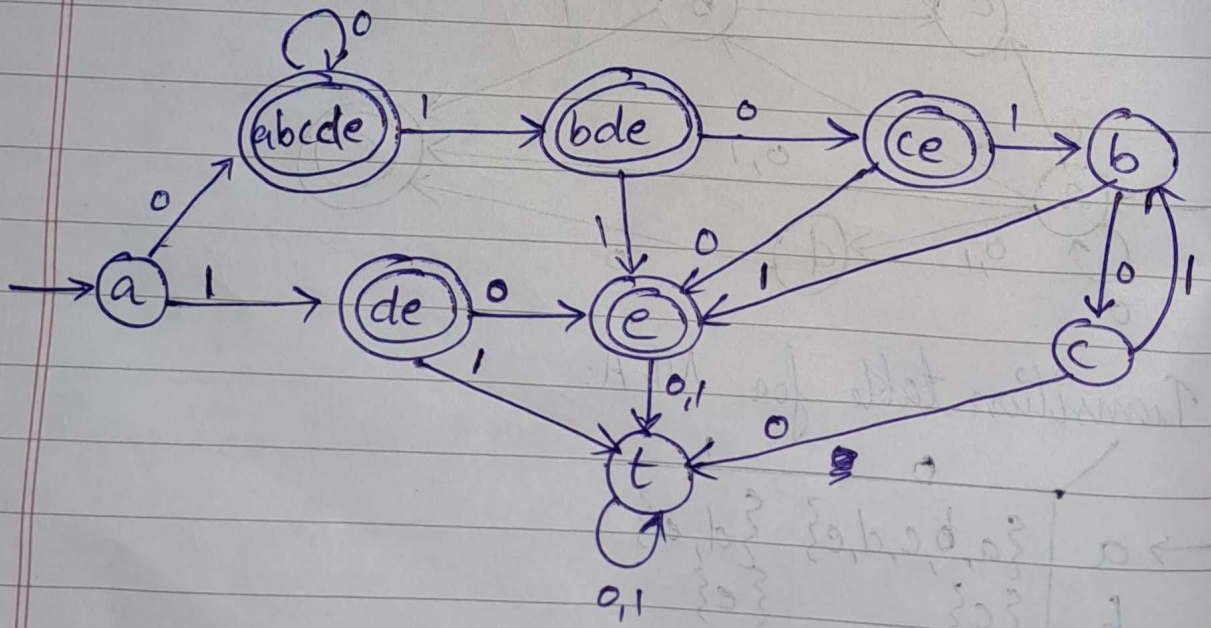
	0	1
→ a	{a,b,c,d,e}	{d,e}
b	{c}	{e}
c	–	{b}
d	{e}	–
* e	–	–

Transition table DFA:

	0	1
→ a	abcde	de
* abcde	abcde	bde
* de	e	t_0
* bde	ce	e
* e	t_1	t_1
t_0	t_0	t_0
* ce	e	b
b	c	e
c	t_2	b
t_2	t_2	t_2

All taap states made into one
that is:: $t_0, t_1, t_2 = t$

Diagram.



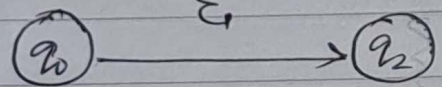
Gayathri B Nair
AM.EN.U4AIE22117

14/10/24/Monday.

ε-NFA

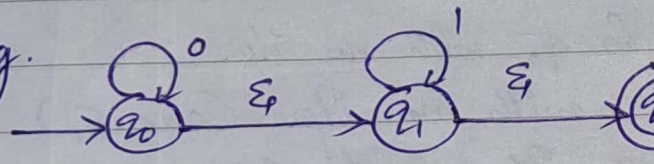
- It is the NFA with ε transitions
- Not possible in DFA.
- without i/p transition takes place.

ε → empty string.



- ε transition.

eg.



$\Sigma = \{0, 1, 2\}$.

1. $\omega = 0$

   $\omega = 0 = \omega = 0\varepsilon\varepsilon$ → so accepted

   So 0 is accepted.

2. $\omega = 00$ → $00\varepsilon\varepsilon$ → accepted.
3. $\omega = 1$ → $\varepsilon 1 \varepsilon$ → accepted.
4. $\omega = 2$ → $\varepsilon\varepsilon 2$ → accepted.

ie; $L_1 = \{0, 00, 000, \ldots, 1, 11, 111, \ldots, 2, 22, 222, \ldots, 01, 02, 12, 0011, 0011 22, \ldots\}$

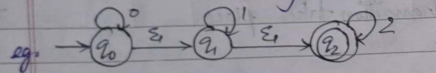
$L_1 = \{\omega \mid \omega$ is of the foam $0^i 1^j 2^k ; i, j, k \geq 0;$

$\Sigma = \{0, 1, 2\}\}$

ε-closure of a state (ε^*)

ε-NFA → NFA → DFA

The set of states that are reachable from the current state using ε-transitions

eg. 

$\Sigma = \{0, 1, 2\}$

$$\varepsilon^*(q_0) = \{q_0, q_1, q_2\}$$

epsilon closure of q_0.

q_0 remains in q_0 with no i/p.
q_0 goes to q_1 with no i/p (one ε)
q_0 goes to q_2 with no i/p (two ε).

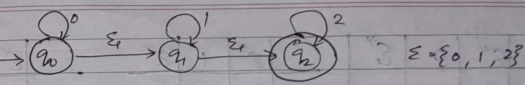
$$\varepsilon^*(q_1) = \{q_1, q_2\}$$

$$\varepsilon^*(q_2) = \{q_2\}$$

Convert ε-NFA to NFA:

Step 1. Find ε^* of all states in ε-NFA.
Step 2. Write the transition table of ε-NFA
Step 3. Find the transition of NFA using the following tabular method

States	ε^*	i/p	ε^*

eg. 

$\Sigma = \{0, 1, 2\}$

Step 1:

$\varepsilon^*(q_0) = \{q_0, q_1, q_2\}$
$\varepsilon^*(q_1) = \{q_1, q_2\}$
$\varepsilon^*(q_2) = \{q_2\}$

Step 2: transition table

	0	1	2	ε
→ q_0	$\{q_0\}$	-	-	$\{q_0, q_1, q_2\}$
q_1	-	$\{q_1\}$	-	$\{q_2\}$
* q_2	-	-	$\{q_2\}$	-

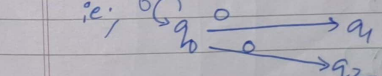
Step 3:

	ε^*	0	ε^* of those states
q_0	q_0	q_0	q_0
	q_1	-	q_1
	q_2	-	q_2

	ε^*	1	ε^*
q_0	q_0	-	-
	q_1	q_1	q_2
	q_2	-	q_2

	ε^*	2	ε^*
q_0	q_0	-	-
	q_1	-	-
	q_2	q_2	q_2

This means that for our NFA, for i/p 0 @ q_0, the transitions are the last column.

i.e. 

NFA	0	1	2
q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
:			

for q_1:

	ε^*	0	ε^*
q_1	q_1	–	–
	q_2	–	–

	ε^*	1	ε^*
q_1	q_1	q_2	q_2
	q_2	–	q_2

	ε^*	2	ε^*
q_1	q_1		
	q_2	q_2	q_2

for q_2

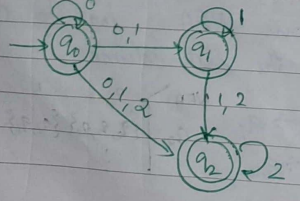
	ε^*	0	ε^*
q_2	q_2	–	–

	ε^*	1	ε^*
q_2	q_2	–	–

	ε^*	2	ε^*
q_2	q_2	q_2	q_2

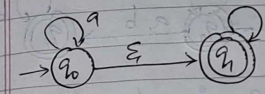
Thus NFA becomes:

	0	1	2
→ * q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
* q_1	–	$\{q_1, q_2\}$	$\{q_2\}$
* q_2	–	–	$\{q_2\}$

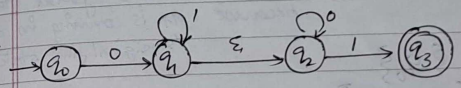


q_0 & q_1 are also final states because the ε^* of q_0 & q_1 has q_2 (the original final state) in it.

Convert the following ε-NFA to NFA:

1. 

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

2. 

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

Answers:

1. $\varepsilon^*(q_0) = \{q_0, q_1\}$
   $\varepsilon^*(q_1) = \{q_1\}$

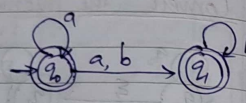
	a	b	ε
→ q_0	$\{q_0\}$	–	$\{q_1\}$
* q_1	–	$\{q_1\}$	–

	ε^*	a	ε^*
q_0	q_0	q_0	q_0
	q_1	–	q_1
q_1	q_1	–	–

	ε^*	b	ε^*
q_0	q_0	–	–
	q_1	q_1	q_1
q_1	q_1	q_1	q_1

NFA transition table:

	a	b
→ * q_0	$\{q_0, q_1\}$	$\{q_1\}$
* q_1	–	$\{q_1\}$



q_0 is also final state because q_1 is coming in ε^* of q_0 ↳ original final state.

2. $\varepsilon^*(q_0) = \{q_0\}$
$\varepsilon^*(q_1) = \{q_1, q_2\}$
$\varepsilon^*(q_2) = \{q_2\}$
$\varepsilon^*(q_3) = \{q_3\}$

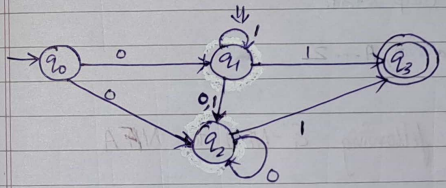
	0	1	ε
→ q_0	$\{q_1\}$	–	–
q_1	–	$\{q_1\}$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_3\}$	–
* q_3	–	–	–

	ε^*	0	ε^*	ε^*	1	ε^*
q_0	q_0	q_1	q_1, q_2	q_0	–	–
q_1	q_1	–	–	q_1	q_1	q_1
	q_2	q_2	q_2		q_3	q_2
						q_3

	ε^*	0	ε^*	ε^*	1	ε^*
q_2	q_2	q_2	q_2	q_2	q_3	q_3
q_3	q_3	–	–	q_3	–	–

Thus NFA transition table is:

	0	1
→ q_0	$\{q_1, q_2\}$	–
q_1	$\{q_2\}$	$\{q_1, q_2, q_3\}$
q_2	$\{q_2\}$	$\{q_2\}$
* q_3	–	–



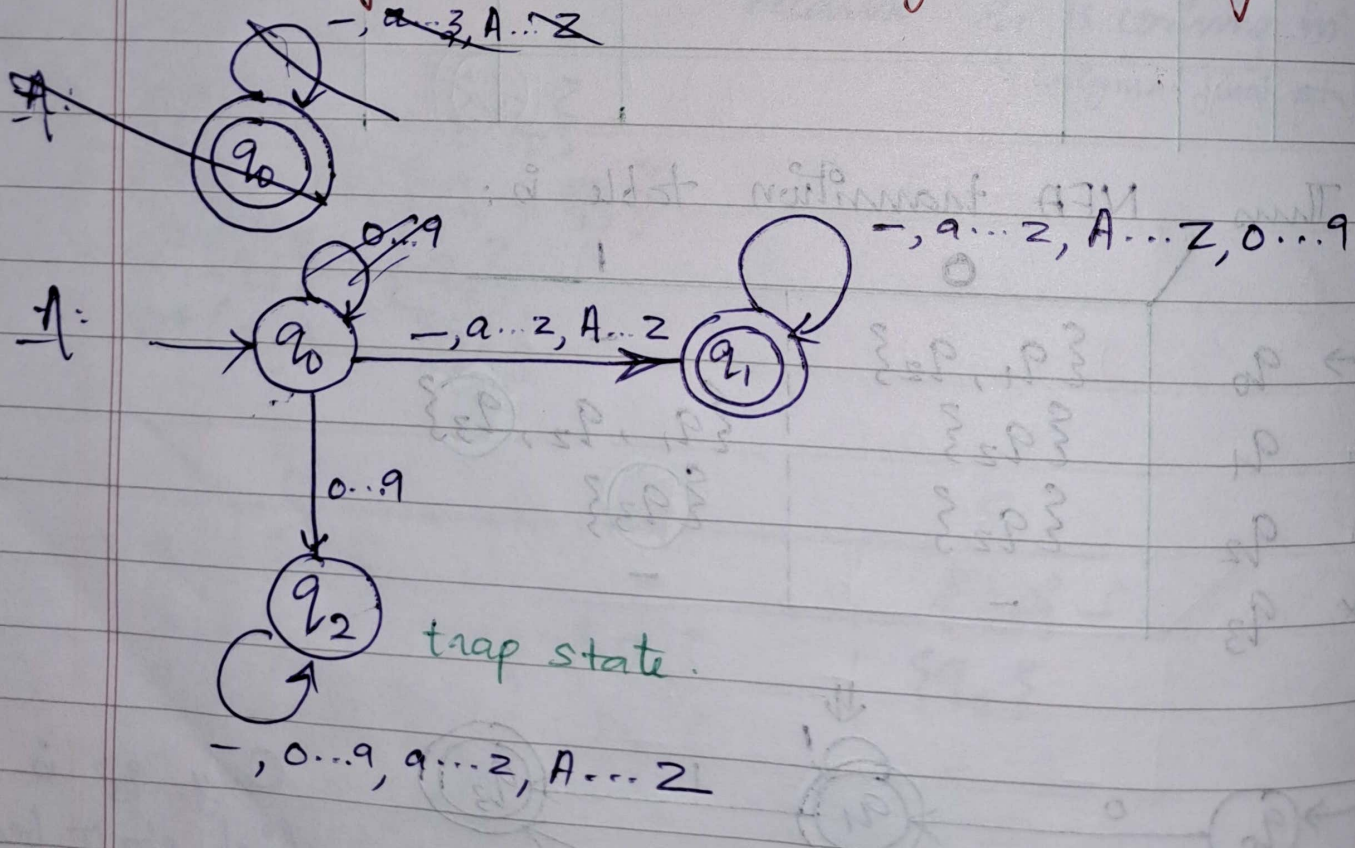
Only q_3 is final state because ε^* of q_0, q_1 & q_2 do not have q_3.

15/10/24/Tuesday.

Q. Design a FA to accept valid identifier

$\Sigma = \{ -, a \cdots z, A \cdots Z, 0 \cdots 9 \}$

rule: if it starts with a digit then reject it.



17/10/24 | Thursday.

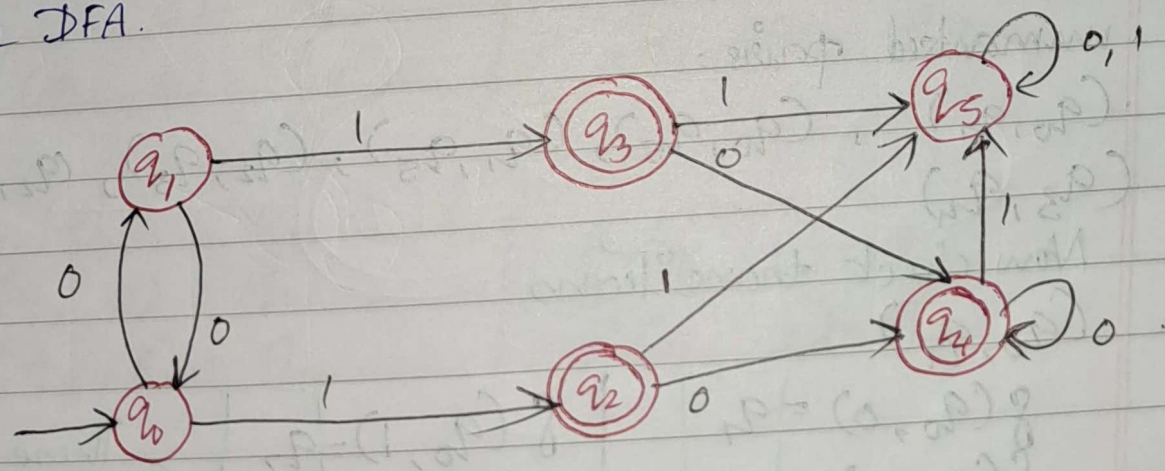
Minimization of DFA using Myhill-Nerode theorem:

- To reduce the no: of states
- Table filling method.

Steps:

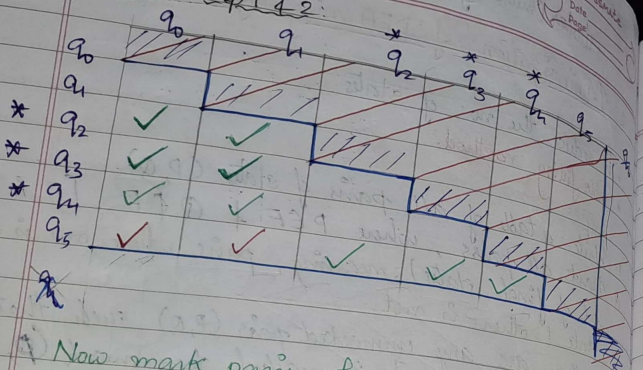
1. Write a table of all pairs of states (P, Q).
2. Mark all pairs, where $P \in F$ & $Q \notin F$ [F is the set of final states]: meaning, pairs where one is a final state & other is not.
3. If there are any unmarked pair (P, Q) such that $[\delta(P, x), \delta(Q, x)]$ is marked then mark (P, Q), where x is the input symbol.
4. Repeat step 3 until no more marking is possible.
5. Combine the unmarked pairs to single unit to reduce the DFA.

eg:



Answer in next page,

Step 1 & 2:



Now mark pairs where one is final state ↑
(q_0, q_3) is also considered even though no direct path is present. (green ticks)

Step 3:

unmarked pairs:

(q_0, q_1), (q_0, q_5), (q_1, q_5), (q_2, q_3), (q_2, q_4), (q_3, q_4)

Now check transitions:

(q_0, q_1)

$\delta(q_0, 0) = q_1$ | $\delta(q_0, 1) = q_2$
$\delta(q_1, 0) = q_0$ | $\delta(q_1, 1) = q_3$

These are (q_0, q_1) / (q_2, q_5) are unmarked so we cannot do anything

(q_0, q_5)

$\delta(q_0, 0) = q_1$ | $\delta(q_0, 1) = q_2$
$\delta(q_5, 0) = q_5$ | $\delta(q_5, 1) = q_5$

(q_1, q_5) + (q_2, q_5)
unmarked ↳ marked → since this is marked we can mark the corresponding pair (q_0, q_5) as well (Red ticks)

(q_1, q_5) — marked it ↑ marked just now

$\delta(q_1, 0) = q_0$ | $\delta(q_1, 1) = q_3$ (marked)
$\delta(q_5, 0) = q_5$ | $\delta(q_5, 1) = q_5$

(q_2, q_3)

$\delta(q_2, 0) = q_4$ | $\delta(q_2, 1) = q_5$
$\delta(q_3, 0) = q_4$ | $\delta(q_3, 1) = q_5$

(q_2, q_4)

$\delta(q_2, 0) = q_4$ | $\delta(q_2, 1) = q_5$
$\delta(q_4, 0) = q_4$ | $\delta(q_4, 1) = q_5$

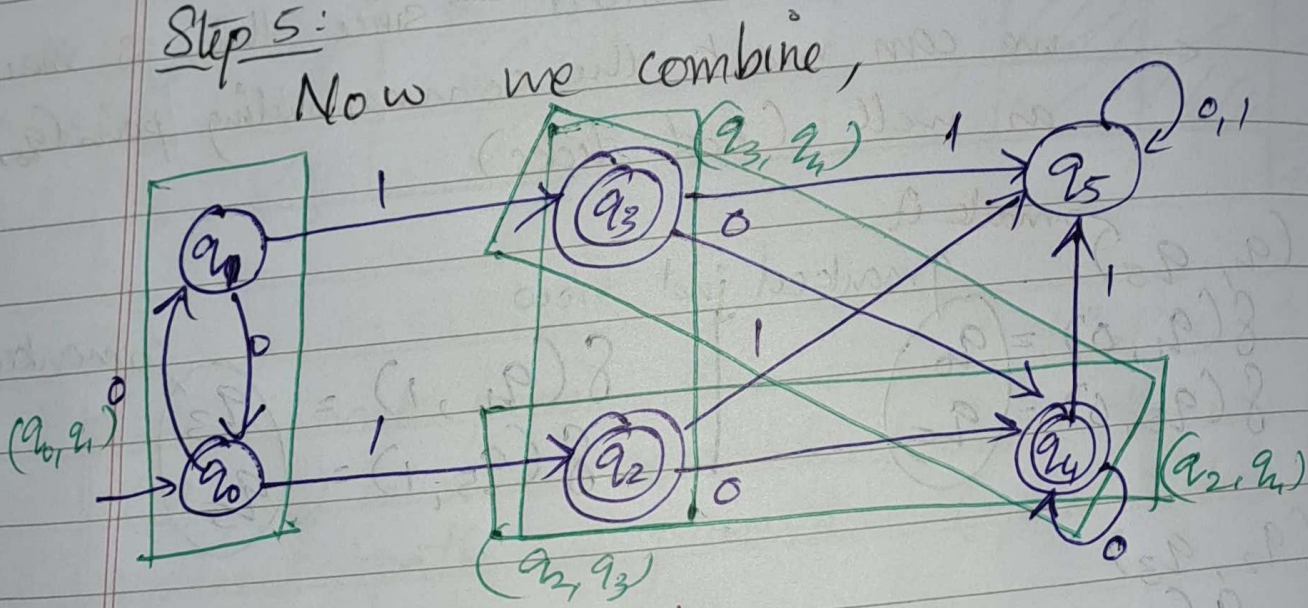
(q_3, q_4)

$\delta(q_3, 0) = q_4$ | $\delta(q_3, 1) = q_5$
$\delta(q_4, 0) = q_4$ | $\delta(q_4, 1) = q_5$

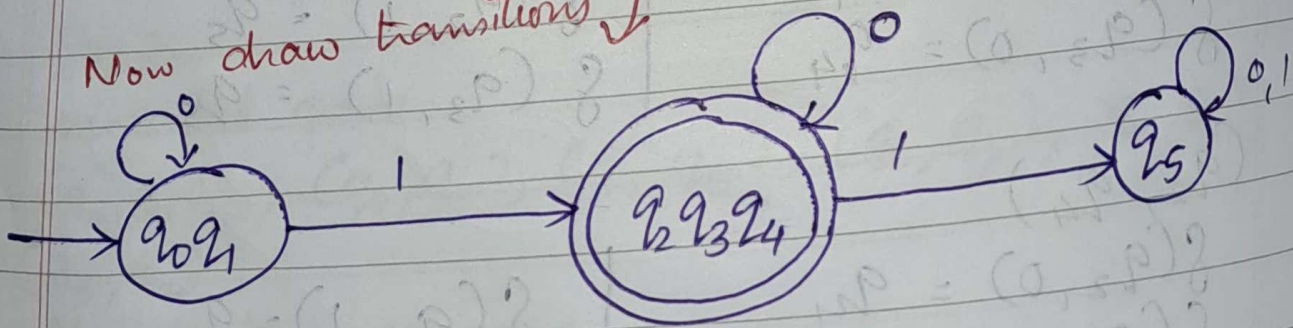
Step 4:
Repeating it again we see 4 unmarked pairs:

(q_0, q_1), (q_2, q_3), (q_2, q_4), (q_3, q_4)

Step 5:
Now we combine,



Now draw transitions ↓



This is the minimized DFA.

21/10/24/Monday.

Regular Expression (RE)

It is an expression that represents a pattern.

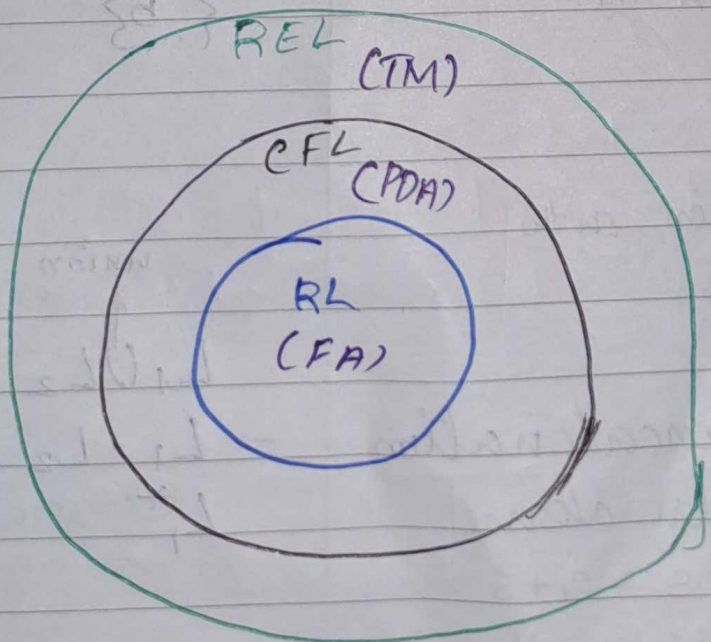
Regular Languages: (RL)
All languages accepted by finite automata.

Context free Languages: (CFL)
All ~~s~~ languages accepted by push down automata

~~Turing Machine~~

Recursively Enumerable Languages: (REL)

All languages accepted by Turing Machine.



ie; Turing machine can accept RL, CFL & REL.

Regular Language

- language accepted by / represented with FA.
- can be represented with regular expressions.

$L_1 \rightarrow$ [FA$_1$] OR RE$_1$

$L_2 \rightarrow$ [FA$_2$] OR RE$_2$

Definition

	RE	Language
1.	ϕ	$\{\}$
2.	ε	$\{\varepsilon\}$
3.	if $\Sigma = \{a, b\}$	
	a → every symbol in alphabet	$\{a\}$
	b → representing a RE.	$\{b\}$

4. if r_1 for L_1
   s for L_2
   we can do operations on the two RE's r & s.
   1. $r + s$ → union → $L_1 \cup L_2$
   2. $r \cdot s$ → concatenation → $L_1 \cdot L_2$
   3. r^* → star closure → L_1 (*) → zero or more.

priority: $r^* > r \cdot s > r + s$

same symbol - then direction: left to right.

say $L_1 = \{a, b, c\}$

$L_2 = \{x, y, z\}$

$r + s$ $L_1 \cup L_2 = \{a, b, c, x, y, z\}$

$r \cdot s$ $L_1 \cdot L_2 = \{ax, ay, az, bx, by, bz, cx, cy, cz\}$

r^* if $L_1 = \{0\}$

$L_1^* = \{0\}^* \rightarrow 0^n \; ; n \geq 0$

↳ zero can occur zero or more times in the string.

$L_1^* = 0^* = \{\varepsilon, 0, 00, 000, \ldots\}$

	RE	Language
1.	0	$\{0\}$
2.	0^*	$\{\varepsilon, 0, 00, 000, \ldots\}$
3.	1^*	$\{\varepsilon, 1, 11, 111, \ldots\}$
4.	$(0+1)$	
	$\{0\} + \{1\}$ → union	$\{0, 1\}$
5.	01	$\{01\}$
6.	~~0~~ $0 \cdot 1^*$	~~$\{0, 01, 0 \cdot 11, 0 \cdot 111, \ldots\}$~~
	$0 \cdot \{\varepsilon, 1, 11, 111, \ldots\}$	$\{0, 01, 011, 0111, \ldots\}$
7.	$1 \cdot 0^* \cdot 1$	$\{11, 101, 1001, \ldots\}$
	$\{1\} \; \{\varepsilon, 0, 00, \ldots\} \; \{1\}$	

$\{\varepsilon, 1, 0\}^*$

8.	$(1+0)^*$	$\{\varepsilon, 0, 1, 00, 01, 10, 11, \ldots\}$
9.	$(0+1)^* \cdot 11$	$\{11, 011, 111, 0011, 0111, \ldots\}$ all strings ending with '11'

Q. Find the regular expression for the following (binary) languages:

1. All strings start with 0.

A: ~~$\{0, 1, 0\}$~~ $0 \cdot (1+0)^*$ → $0 \cdot (1+0)^*$

2. All strings end with 00

A: $(1+0)^* \cdot 0 \cdot 0$ → $(1+0)^* \cdot 00$

3. All strings containing 101

A: $(0+1)^* \cdot 1 \cdot 0 \cdot 1 \cdot (0+1)^*$ → $(0+1)^* \cdot 101 \cdot (0+1)^*$

4. all strings containing three 1s

A: ~~$(0+1)^* \; 0^* \; 1 \; 1 \; 1 \; 0^*$~~

~~$0^* \; 1 \; 0^* \; 1 \; 0^* \; 1 \; 0^*$~~

$(0+1)^* \cdot 1 \cdot (0+1)^* \cdot 1 \cdot (0+1)^* \cdot 1 \cdot (0+1)^*$

5. All strings containing odd no. of 0s

A: ~~$1^* \cdot 0 \cdot 1^* (1^* 0 1^* 0 1^*)^* \cdot 1^*$~~

$1^* \cdot 0 \cdot 1^* \cdot (1^* \cdot 0 \cdot 1^* \cdot 0 \cdot 1^*)^* \cdot 1^*$

6. All strings containing exactly ~~3 ones?~~ three 1s?

A: ~~$(0+1)^* \cdot 111 \cdot (0+1)^*$~~

$0^* \cdot 1 \cdot 0^* \cdot 1 \cdot 0^* \cdot 1 \cdot 0^*$

6. All strings containing exactly ~~3 ones~~ three 1s?

A: ~~$(0+1)^* \cdot 111 \cdot (0+1)^*$~~

$0^* \cdot 1 \cdot 0^* \cdot 1 \cdot 0^* \cdot 1 \cdot 0^*$

22/10/24 | Tuesday.

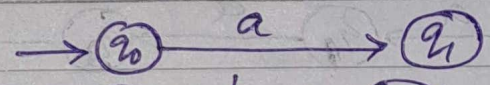
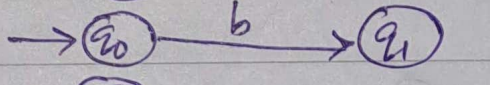
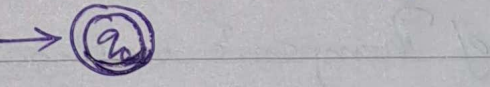
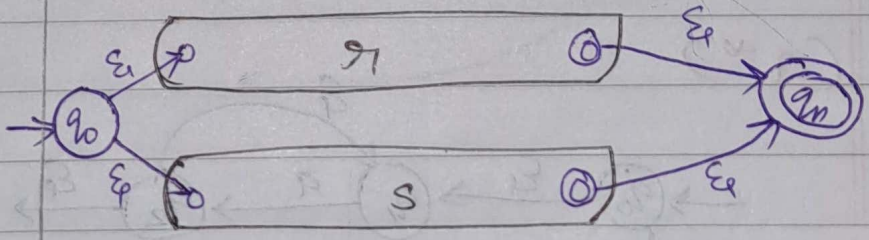
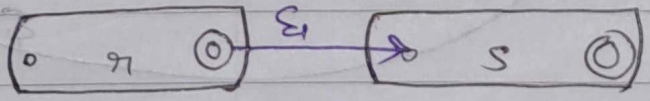
7. {All strings start with 1 and end with 00} $\Sigma = \{0, 1\}$

A:

$1 \cdot (0+1)^* \cdot 00$

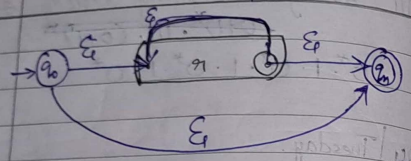
RE to FA : (Thompson's Model)

RE $\leftrightarrow$ FA

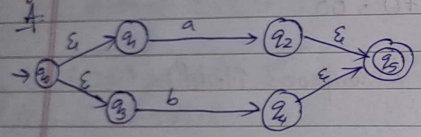
RE	FA
a	
b	
ε	
$r+s$	
$r \cdot s$	

RE → FA

r^*

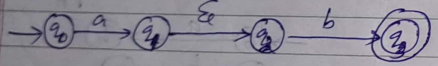


eg. Draw FA of (a+b) where $\Sigma = \{a, b\}$



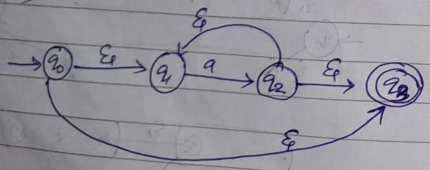
$\{a, b\}$

(a.b)



* Disadv of Thompson's method: too many ε

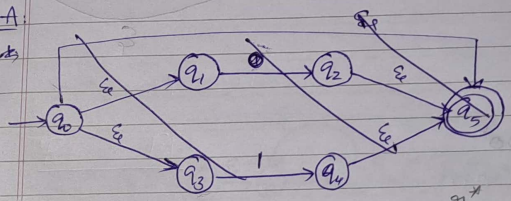
(a^*)



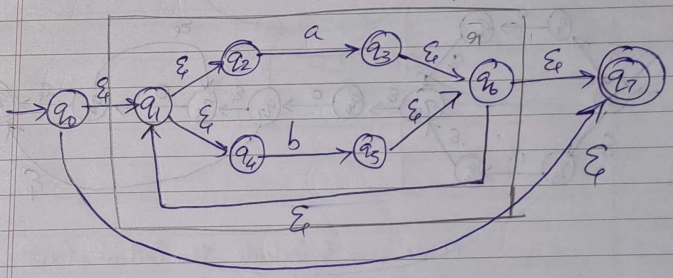
Q. Draw FAs for following REs?

1. $(1+0)^*$
2. 001^*
3. $(0+1)00^*$
4. $01(0+1)01$
5. 01^*010^*

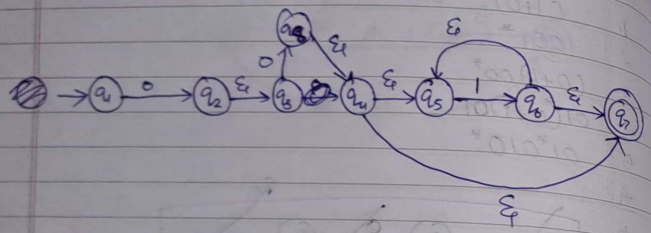
A:



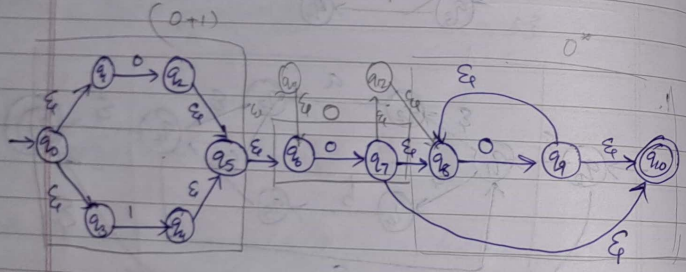
1.



2. 001^*

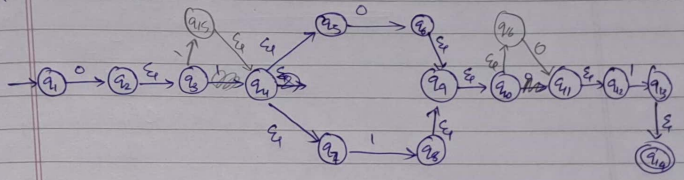


3. $(0+1)00^*$

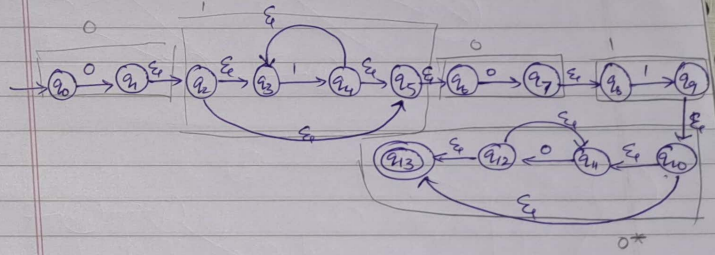


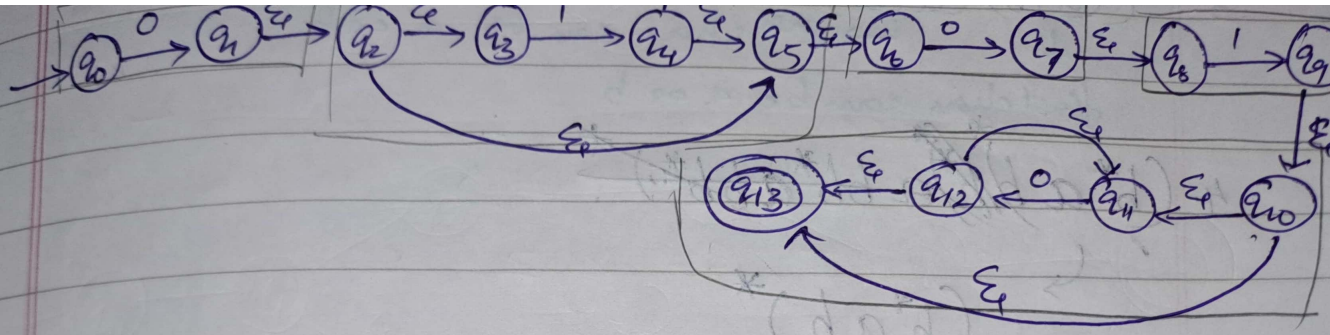
Make sure to include the initial states of all individual parts.

4. $01(0+1)01$



5. 01^*010^*





24/10/24 / Thursday.

Q. Write RE for the following languages.

All strings

1. containing exactly 2 b's $\Sigma = \{a, b\}$
2. starts with 'ab', ends with 'ba' $\Sigma = \{a, b\}$
3. with even length. $\Sigma = \{a, b\}$
4. contains no consecutive a's. $\Sigma = \{a, b\}$
5. ~~contains~~ with alternating 0's and 1's $\Sigma = \{0, 1\}$

6. contains atleast one 1. $\Sigma = \{0, 1\}$
7. contains exactly two 0's. $\Sigma = \{0, 1\}$
8. atleast two a and three b $\Sigma = \{a, b\}$
9. first and last character are the same $\Sigma = \{a, b\}$
10. First & last characters ~~are~~ should not be same. $\Sigma = \{a, b\}$

Answers:

1. $a^* \cdot b \cdot a^* \cdot b \cdot a^*$
2. $ab\,(a+b)^*ba$
3. ~~$((a+b)^*(a+b)^*)^*$~~
3. $(a+b)(a+b)^*$ → repeated 0 or more.
   ↳ 2nd char can be a or b
   ↓
   first char can be a or b
4. ~~$(b^*ab^*a^*bb^*abb^*)^*$~~
   ↳ $(b^*ab)^*$ $(b+ab)^*$
5. ~~$(01+10)^*$~~ ~~$0+0$~~
5. $(01)^* + (10)^*$ $(01)^*$
6. ~~0^*1^*0~~ $X0^*1^*0^*1^*$ $(0+1)^*1(0+1)^*$
7. $1^* \cdot 0 \cdot 1^* \cdot 0 \cdot 1^*$
8. $(a+b)^*a(a+b)^*a(a+b)^*b(a+b)^*b(a+b)^*b(a+b)^*$
9. $(a+b) + a(a+b)^*a + b(a+b)^*b + \varepsilon$
10. $(a+b)^*ab(a+b)^*ab(a+b)^*$

Regular Languages:

- A language is said to be regular if there exists any finite automata (DFA / NFA) it accept the language.
- represented using FA or RE.

Properties

- Regular languages are closed under the following properties:
  ↳ meaning if L_1 & L_2 → RL then $L_1 \cup L_2$ → RL
1. Union
2. Concatenation
3. Star Closure / Kleene Closure
4. Intersection
5. Reverse
6. Complement

↓ If the op: is done on two RL, then the op is also an RL.

Let 'L' be a regular language, there exists a constant 'p' called pumping length which depends on the language L such that any string s in L with |s| ≥ p can be divided into 3 parts s = xyz satisfy the following conditions:

1. Length condition
   $|xy| \leq p$
   This ensures that x & y together are within the first p characters of s.

2. Non-empty condition
   $|y| > 0$
   This ensures that y is not empty, that means the path that we pump has some content.

3. Pumping Condition
   for any $i \geq 0$, $xy^iz \in L$
   Even if we repeat y, i no: of times, it still belongs to the language.

We cannot count or compare the no: of letters in FA.

Q. $L = \{ a^n b^n \mid n \geq 0 \}$. Prove whether this is a RL or not?

A. Proof:

p - pumping length

s - $\underbrace{aa\cdots aa}_{n}\underbrace{bb\cdots b}_{n}$

Divide s into x y & z

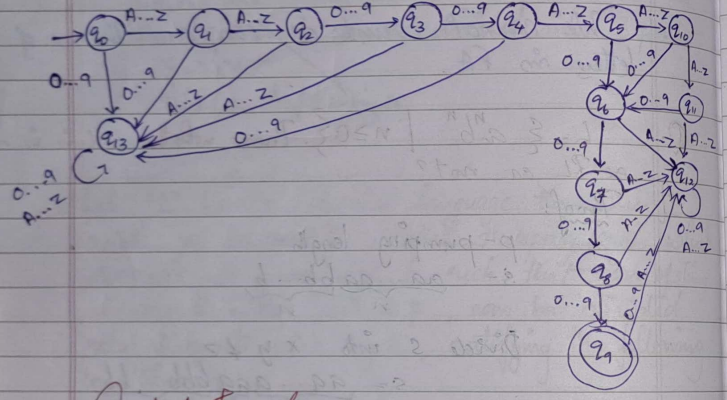
s = $\underbrace{aa\cdots a}_{x}\underbrace{aa}_{y}\underbrace{bbb\cdots bb}_{z}$

Since xy^iz is not in L, L is not RL.

12/11/24/Tue.

MidSem Answer

1. 2 letters + 2 digits + 1/2/3 letters + 4 digits



Context Free Grammar:

- Grammar: In this context grammar is the set of rules to make words/strings using the alphabet.
  production rules. Rules to frame strings in language.
- The language made using this is Context Free Language
  ↓
  represented using push down automata.

Format:

A → BC ; A can be exchanged with BC
A → a ; A can be replaced with a
S → Aa ; S can be replaced with Aa

single variables.

capital letters → single variables.
small letters → terminals / constants

In every rule you MUST ONLY HAVE single variables. This is because: single variables are replaceable and terminals/constants are not replaceable.

These rules are called productions.

Definition for CFG: Context Free Grammar is defined as a 4-tuple (V, T, P, S)

- V → finite set of variables (all capital letters)
- T → finite set of terminals (all small letters)
- P → production rules
- S → symbol where the production/derivation of string starts

eg: P = { S → AB
A → a
B → b }

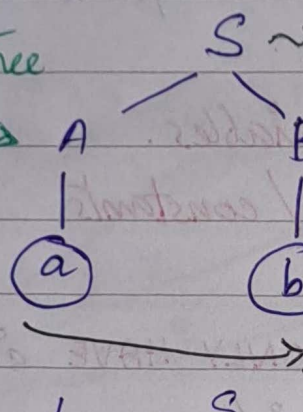
Now we are going to derive some strings from these ~~rule~~ production rules.

eg1: $V = \{S, A, B\}$

$T = \{a, b\}$

We are going to derive using a tree structure.

① Derivation tree or Parse tree.



S → start of derivation.

variable so we need to continue.

We reach terminal, we stop

read from left to right (leaf nodes) and you get the string.

Thus $L_1 = \{ab\}$

We use these trees in compilers.

Another Method:

② Sentential Form:

$S \Rightarrow AB$

$\Rightarrow aB$

$\Rightarrow ab$ Thus we get $L_1 = \{ab\}$

eg2:

14/11/24/ Thursday.

Terminologies

1. **Non Terminal / Variable (V)**
   Variables are the symbols that can be replaced by other symbols

2. **Terminal (T):**
   They are the symbols that cannot be replaced.

3. **Start Symbol (S):**
   It is a non-terminal or variable where the derivation of a string starts.

4. **Production Rule (P):**
   It is the grammar rule of the form
   $A \rightarrow \alpha$ where α - terminal / non-terminals / a combination of both.

Grammar → G = (V, T, P, S)

We give the grammar to the parser.

Eg2. Derive the expression:

$G = (V, T, P, S)$

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

$T = \{a, b\}$

$P = \{ S \to AB$

$A \to aaA$

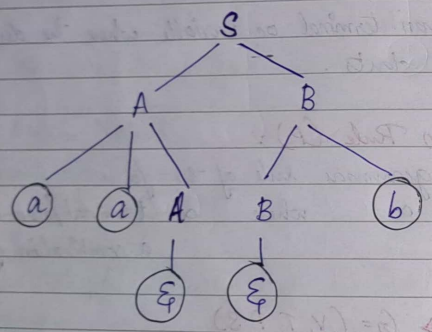
$A \to \xi$

$B \to Bb$

$B \to \xi \}$

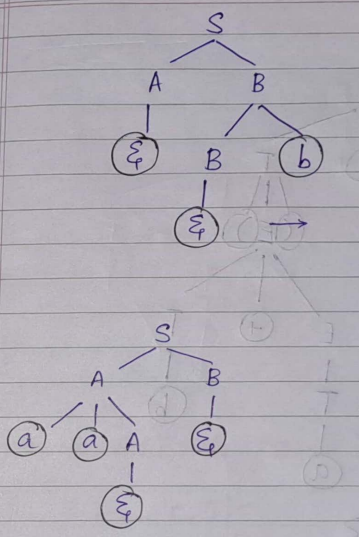
Find one or two strings belonging to this language.

A:



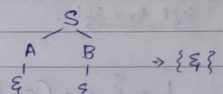
→ left to right (read leaf nodes)

$L = \{ aa\xi\xi b \} = \{aab\}$



Thus another string is: $\xi\xi b = b$.

Another string: $aa\xi\xi = aa$



→ $\{\xi\xi\}$

There can infinitely many strings. Some of them include $L = \{aab, b, aa\xi, \xi\}$

↳ of the form $a^n b^m$ when $n \geq 0$ $m \geq 0$

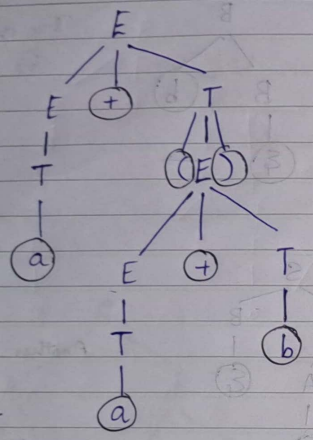
Q. Write a grammar for addition operation?

A: $G = (V, T, S, P)$ $S = \{E\}$ → only

$V = \{E, T\}$ $P = \{ E \to E+T \mid T\xi$ → $E \to E+T$, $E \to T$, $T \to (E)$, $T \to a$, $T \to b$

$T = \{a, b\xi, +, (,)\}$ $T \to (E) \mid a \mid b \}$

eg: a+(a+b)



left to right.

$L = \{ a+(a+b) \}$

Order Of Derivation:

1. Left Most Derivation - LMD
2. Right Most Derivation - RMD

This is the order of derivation or in which order we replace variables.

LMD

$E \Rightarrow E+T$
$\Rightarrow T+T$
$\Rightarrow a+T$
$\Rightarrow a+(E)$
$\Rightarrow a+(E+T)$
$\Rightarrow a+(T+T)$
$\Rightarrow a+(a+T)$
$\Rightarrow a+(a+b)$

always replace the leftmost variable first.

RMD

$E \Rightarrow E+T$
$\Rightarrow E+(E)$
$\Rightarrow E+(E+T)$
$\Rightarrow E+(E+b)$
$\Rightarrow E+(T+b)$
$\Rightarrow E+(a+b)$
$\Rightarrow T+(a+b)$
$\Rightarrow a+(a+b)$

always replace the rightmost variable first.

Q. Check whether the following strings follow the grammar given.

$P = \{ E \rightarrow E+T \mid T$
$T \rightarrow T*F \mid F$
$F \rightarrow (E) \mid a \mid b \mid c \}$

$w_1 = a+b*c$
$w_2 = a+b*$

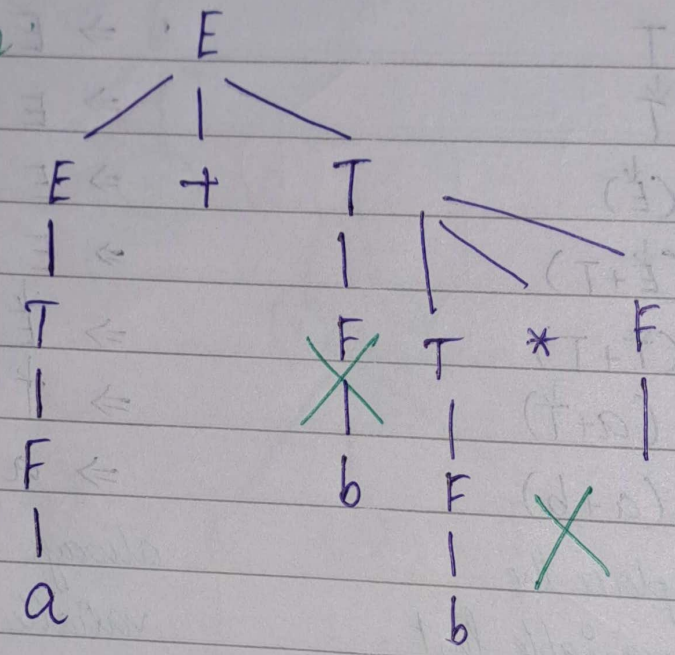
A: $w_1 = a+b*c$

$E \Rightarrow E+T$
$\Rightarrow T+T$
$\Rightarrow F+T$
$\Rightarrow a+T$
$\Rightarrow a+T*F$
$\Rightarrow a+F*F$
$\Rightarrow a+b*F$
$\Rightarrow a+b*c$

That is, it follows the grammar (used Sentential form)

$\alpha_2 = a+b*$

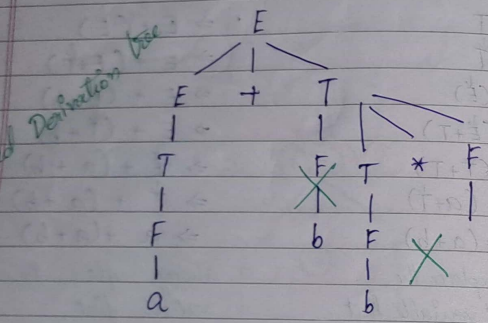
Used Derivation Tree



Thus it cannot be made. That is it is not a part of the language represented by the grammar.

$\omega_2 = a+b*$

Used Derivation tree



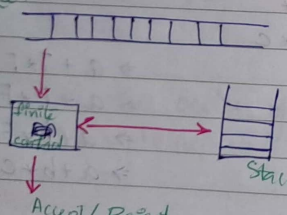
Thus it cannot be made. That is it is not a part of the language represented by the grammar.

18/11/24/Monday.

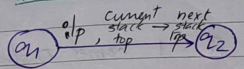
Push Down Automata

- PDA → FA + Stack (→ LIFO)
  → Context Free Language.

Architecture

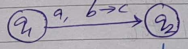


Transition

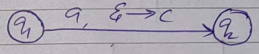


Values written on transition:
- i/p.
- current stack top element
- next stack top element

eg:



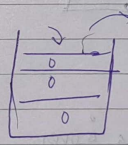
If we want to put push only operation:



Q $L = \{ 0^n 1^n \mid n \geq 1 \}$

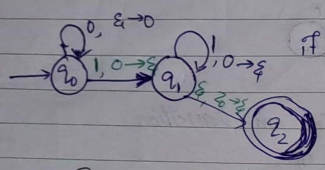
Stack → empty → accepted

$\omega = 000111$



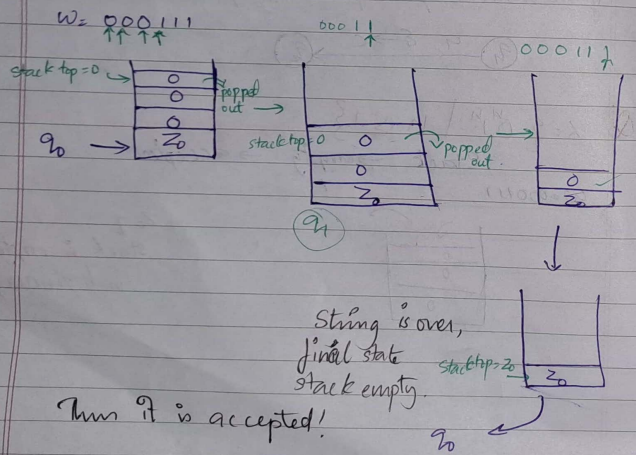
Z_0 → stack is empty & stack is in beginner phase.

Machine:



If 1 comes 0 should be there be 1 as stack top

To accept the ip:
- → end of string
- → stack empty
- → final state



Thus it is accepted!

The intermediate states of the PDA can be marked as Instantaneous Description (ID).
Represented as 3 element tuple:
(current state, remaining input, stack content)

The w_0 = 000111 in the machine traversal will be:

$(q_0, \underline{0}00111, Z_0) \vdash (q_0, \underline{0}0111, 0Z_0)$

$\vdash (q_0, 0111, 00Z_0)$

$\vdash (q_0, \underline{1}11, 000Z_0)$

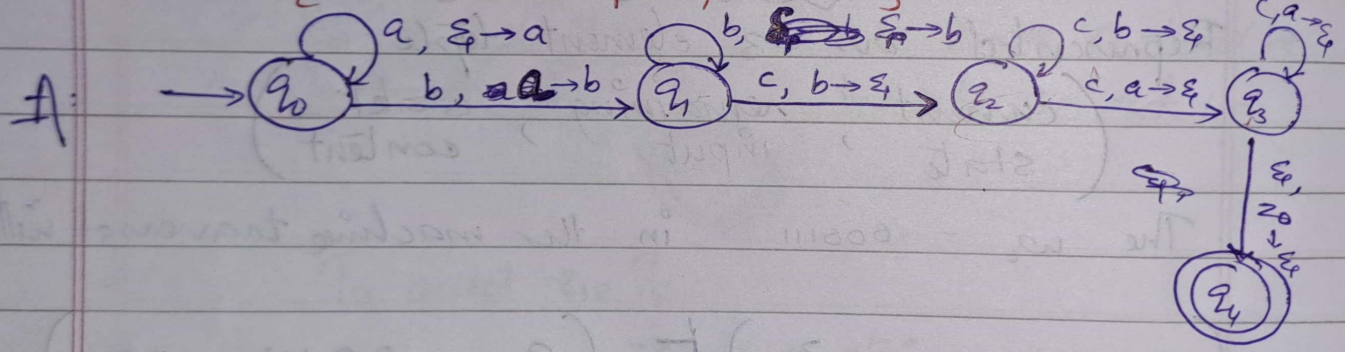
$\vdash (q_1, \underline{1}1, 00Z_0)$

$\vdash (q_1, \underline{1}, 0Z_0)$

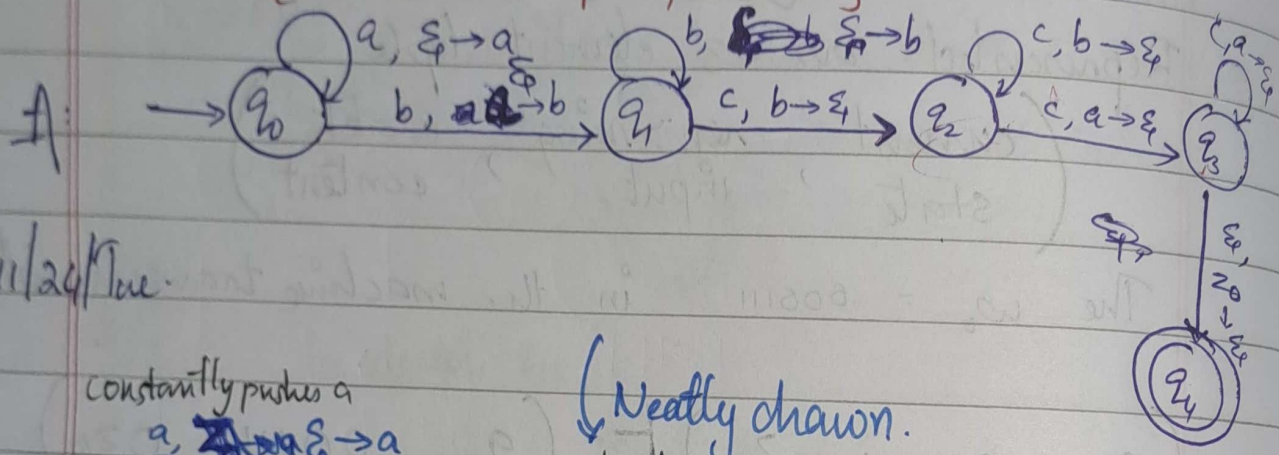
$\vdash (q_1, \varepsilon, Z_0)$

$\vdash (q_2, \varepsilon, \varepsilon)$ Accept

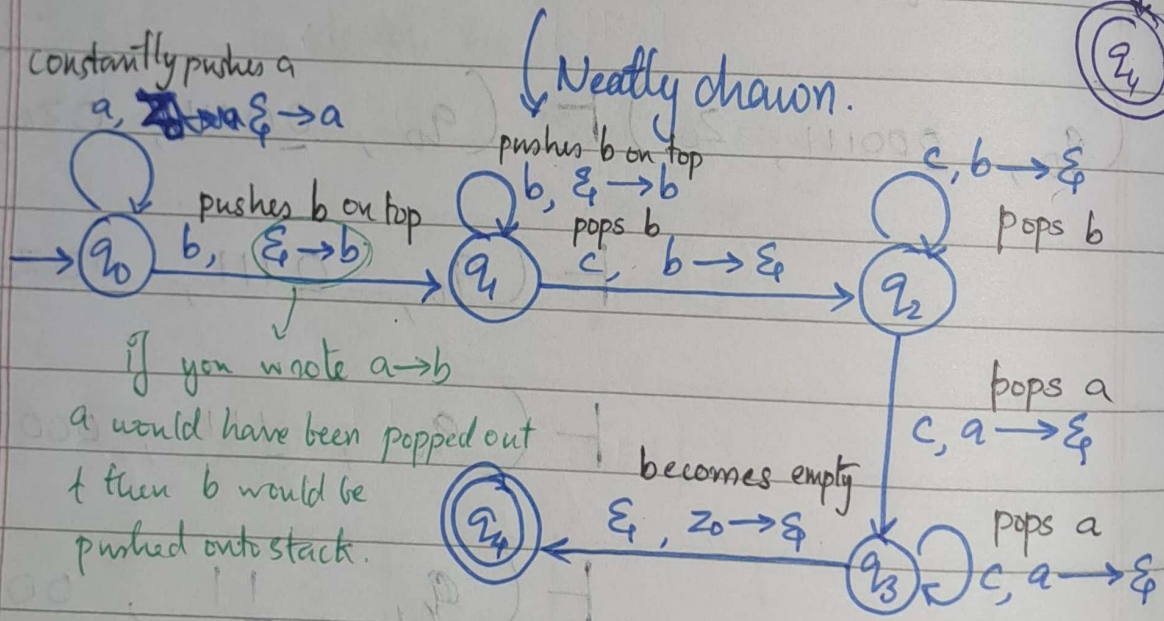
① Design a PDA to accept $L = \{a^n b^m c^{m+n} \mid m, n > 0\}$



Q. Design a PDA to accept ~~$L = \{a^n b^m c^{m+n}\}$~~ $L = \{a^n b^m c^{m+n} \mid m, n > 0\}$



19/11/24 Tue.



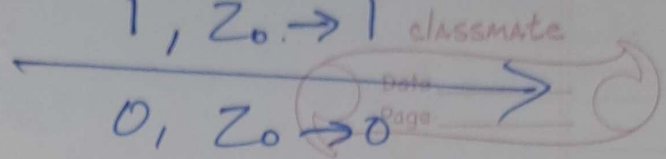
Q. Design PDA for the following languages:

1. $L_1 = \{wcw^R \mid w \in \{0,1\}^*\}$

eg: 011c110

2. $L_2 = \{ww^R \mid w \in \{0,1\}^*\}$

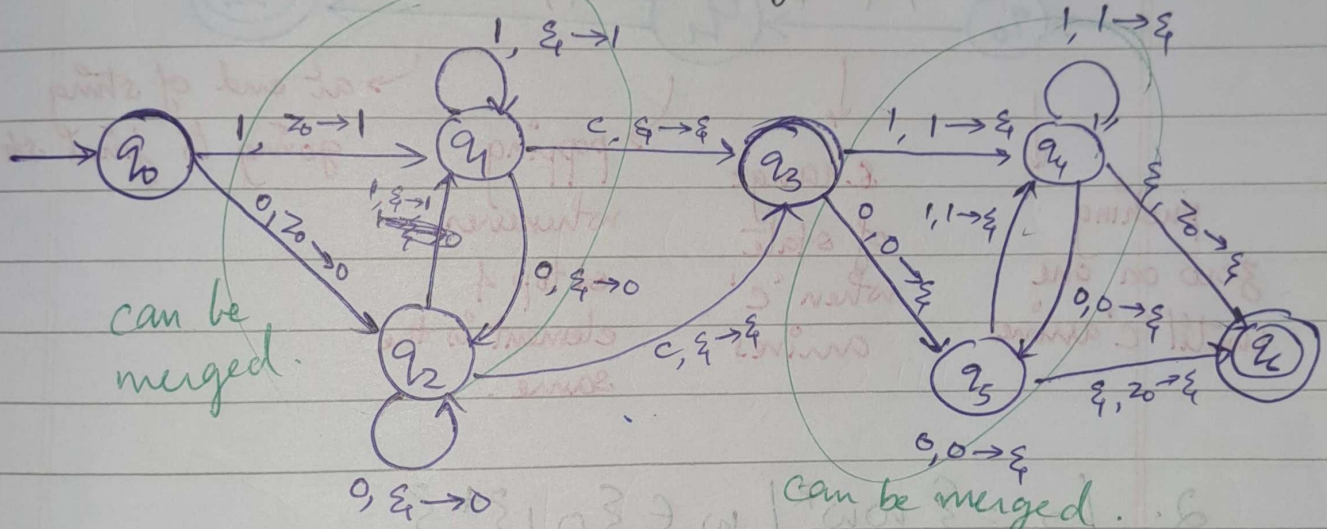
3. ~~$L_3 = \{a^n b^{2n} \mid n \geq 0\}$~~



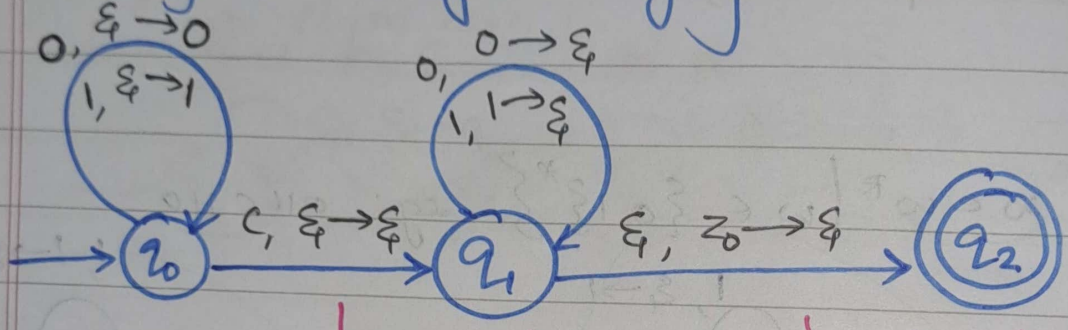
3. $L_3 = \{a^n b^{2n} \mid n \geq 0\}$

Ans

1. $L_4 = \{w c w^R \mid w \in \{0,1\}^*\}$ eg: 011c110



Neatly drawn after merging:



pushing zero on one until 'c' arrives

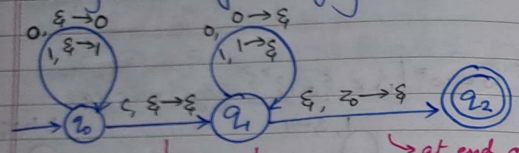
change of state when 'c' arrives

popping whenever at top & element is the same.

at end of string going to final state.

2. $L_2 = \{ ww^R / w \in \{0,1\}^* \}$

Neatly drawn after merging:



pushing zero on one until 'c' arrives

change of state when 'c' arrives

popping whenever top 4 element is the same.

↳ at end of string going to final state.

2. $L_2 = \{ ww^R \mid w \in \{0,1\}^+ \}$

Later: will be continued...

28/11/24 Thursday

Grammars

Hierarchy of Grammars (Chomsky):

1. Type 0 - unrestricted grammar.
2. Type 1 - context sensitive grammar
3. Type 2 - context free grammar
4. Type 3 - Regular Grammar.

★ Regular Language (Type 3 grammar) — acceptor → Finite Automata; generator → Regular Grammar; representation → Regular Expression.

★ Context Free Language (Type 2 Grammar) — acceptor → Push down Automata

★ Context Sensitive Grammar (Type 1) — machine/acceptor ~~language~~ → LBA (Linear Bounded Automata (special Turing Machine))

★ Unrestricted Grammar (Type 0) — language → Recursively Enumerable Language (REL) — acceptor → Turing Machine.

↳ How to recognise regular grammar?

Format:

$A \longrightarrow Bx$ (left) or $A \longrightarrow x$ — Left Linear Grammar.

or

$A \longrightarrow xB$ (right) or $A \longrightarrow x$ — Right Linear Grammar.

$x \rightarrow$ string of terminals.

That is: either full terminals or one terminal variable & a terminal.

Q. Check whether the following are in regular grammar?

1. $\{ S \rightarrow aB$
   $S \rightarrow \varepsilon \}$

A: Right Linear Grammar; Regular Grammar.

2. $\{ S \rightarrow aAb$
   $S \rightarrow \varepsilon$
   $A \rightarrow Aa \}$

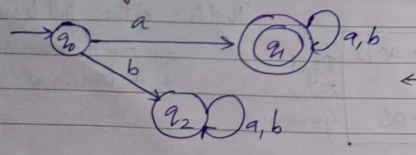
A: Not a regular grammar.

3. $\{ S \rightarrow aaA \mid Bbb \mid \varepsilon$
   $A \rightarrow aA$
   $B \rightarrow Bb \}$

A: Cannot be categorised as either left or right linear. In regular grammar all variables must come in the same side. So this is just the ~~t~~ Linear Grammar and NOT Regular Grammar.

Q. Say a regular language: $L = \{ w \mid w$ starts with 'a', $\Sigma = \{a, b\} \}$. Make the grammar for it.

A:



← The DFA

Grammar: ~~$G(V, T, P, S)$~~ $G = (V, T, P, S)$

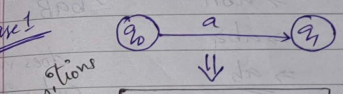
$V = \{ q_0, q_1, q_2 \}$

$T = \{ a, b \}$

S = initial state (q_0)

Production:

case 1 transitions



case 2 final states

final states q

$q \rightarrow \varepsilon$

The rules.

Now going on to the grammar:

$$P = \{ \begin{array}{l} q_0 \rightarrow aq_1 \\ q_1 \rightarrow aq_1 \mid bq_1 \\ q_1 \rightarrow \varepsilon \\ q_0 \rightarrow bq_2 \\ q_2 \rightarrow aq_2 \mid bq_2 \} \end{array}$$

$$P = \{ \begin{array}{l} q_0 \rightarrow aq_1 \mid bq_2 \\ q_1 \rightarrow aq_1 \mid bq_1 \mid \varepsilon \\ q_2 \rightarrow aq_2 \mid bq_2 \} \end{array}$$

$q_0 \to S$ $q_1 \to A$ $q_2 \to B$ → automata

Now generate strings :-

$\{ S \to aA \mid bB$
$A \to aA \mid bA \mid \xi$
$B \to aB \mid bB \}$

$S \Rightarrow aA$
$\Rightarrow aaA$
$\Rightarrow aabA$
$\Rightarrow aab\xi$
$\Rightarrow aab$

$S \Rightarrow aA$
$\Rightarrow a\xi$
$\Rightarrow a$

$S \Rightarrow aA$
$\Rightarrow abA$
$\Rightarrow ab\xi$
$\Rightarrow ab$

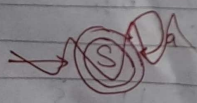
$S \Rightarrow bB$
$\Rightarrow baB$
⋮
does not end with a terminal & thus not accepted.

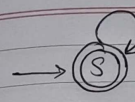
↓

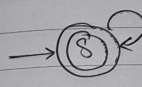
such productions are called "useless productions"

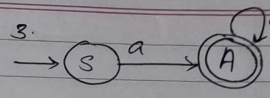
Here $S \to bB$ is a useless production.

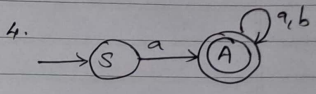
Q2 Write the production rules for the following



1. 

2. 

3. 

4. 

Ans: 1. $P = \{ S \to aS \mid \xi \}$ → $L = \{ w \mid w = a^n ; n \geq 0 \}$

2. $P = \{ S \to aS \mid bS \mid \xi \}$ → $L = \{ w \mid w = a^m b^n ; m \geq 0, n \geq 0 \}$

3. $P = \{ S \to aA$
$A \to aA \mid \xi \}$ → $L = \{ w \mid w = a^n ; n \geq 1 \}$

4. $P = \{ S \to aA$
$A \to aA \mid bA \mid \xi \}$ → $L = \{ w \mid w = a^m b^n ; m \geq 1, n \geq 0 \}$

A: 1. $P = \{ S \longrightarrow aS \mid \epsilon \}$ $\rightarrow L = \{ w \mid w = a^n ; n \geq 0 \}$

2. $P = \{ S \longrightarrow aS \mid bS \mid \epsilon \}$ $\rightarrow L = \{ w \mid w = a^m b^n ; m \geq 0, n \geq 0 \}$

3. $P = \{ S \longrightarrow aA$
$A \longrightarrow aA \mid \epsilon \}$ $\rightarrow L = \{ w \mid w = a^n ; n \geq 1 \}$

4. $P = \{ S \longrightarrow aA$
$A \longrightarrow aA \mid bA \mid \epsilon \}$ $\rightarrow L = \{ w \mid w = a^m b^n ; m \geq 1, n \geq 0 \}$

30/11/24/Saturday.

Tutorial -3 (PDA):

Design PDA for the following languages.

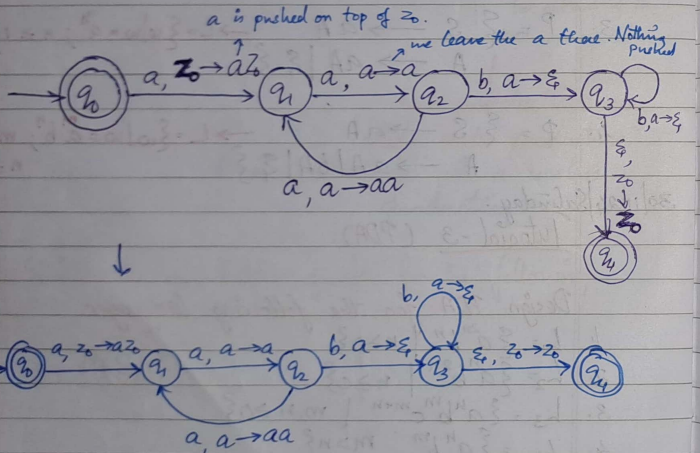
1. $L_1 = \{ a^{2n} b^n \mid n \geq 0 \}$
2. $L_2 = \{ a^n b^{2n} \mid n \geq 0 \}$
3. $L_3 = \{ a^n b^m c^{m+n} \mid m, n \geq 0 \}$
4. $L_4 = \{ a^n b^m , m > n \}$
5. $L_5 = \{ a^{m+n} b^m c^n , m, n \geq 0 \}$

Answers:

1. $L_1 = \{\varepsilon, aab, aaaabb, aaaaaabbb, \ldots\}$

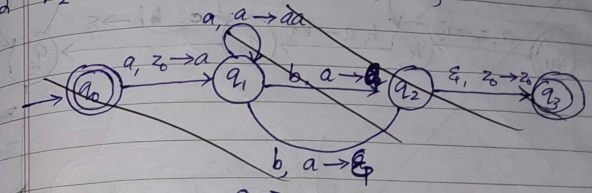
Logic1: push every 2nd a.
pop a whenever you get a b.

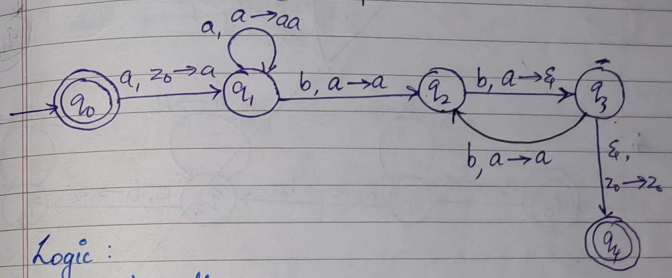
Logic2: push all a.
Whenever a b occurs pop 2 as.



pushes odd positioned as and leaves the stack untouched whenever even positioned as occur. Pop an a whenever b occurs. Repeat until end of string.

2. $L_2 = \{\varepsilon, abb, aabbbb, \ldots\}$





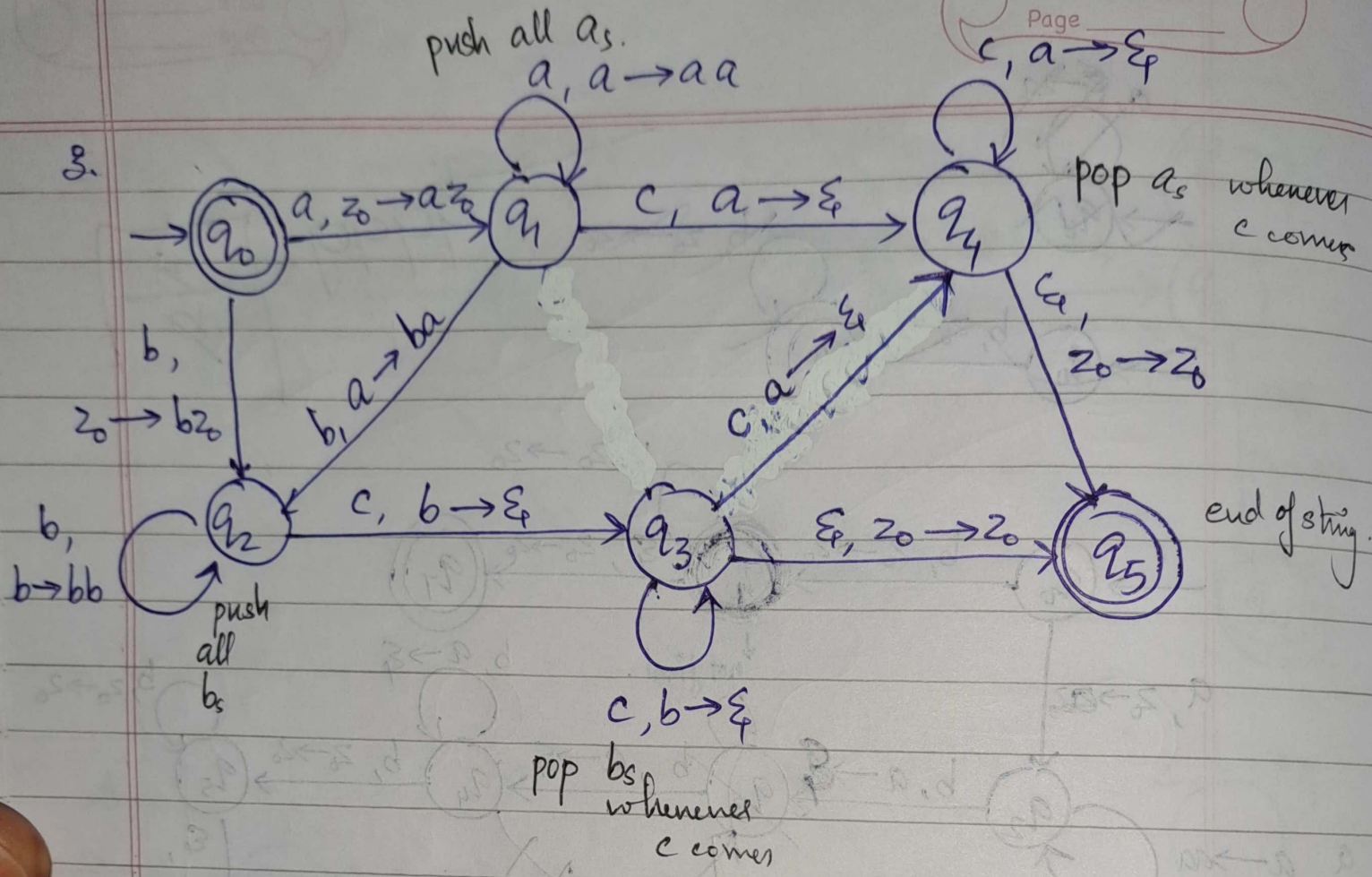
Logic:
push all as.
When b occurs first time, leave the a at top.
Next time occurs pop a.
Repeat until stack empty. ~~Reaches~~ Reaches end of string ⇒ which is accepted.

3. $L_3 = \{\varepsilon, abcc, aabccc, acc, ab, \ldots$

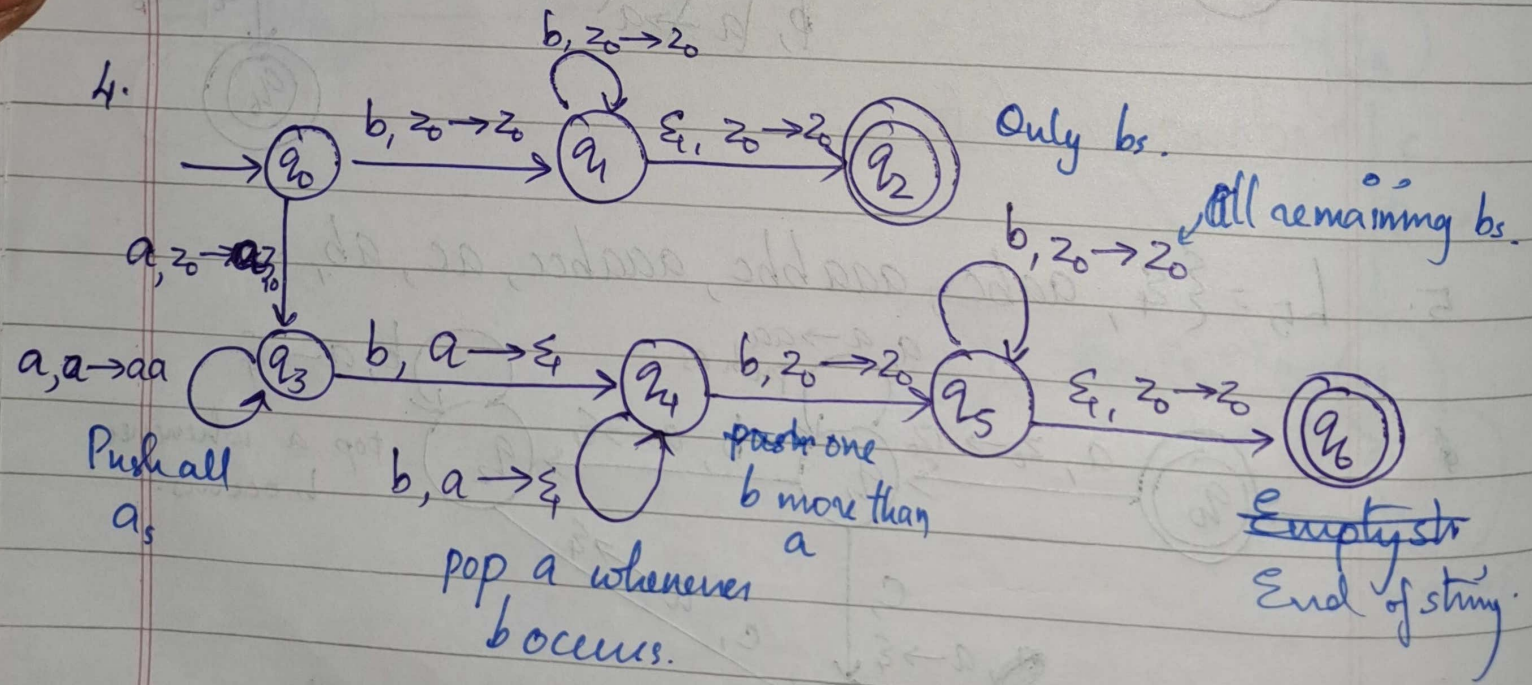
Initial state is final. Pop whenever c occurs.

The Logic.

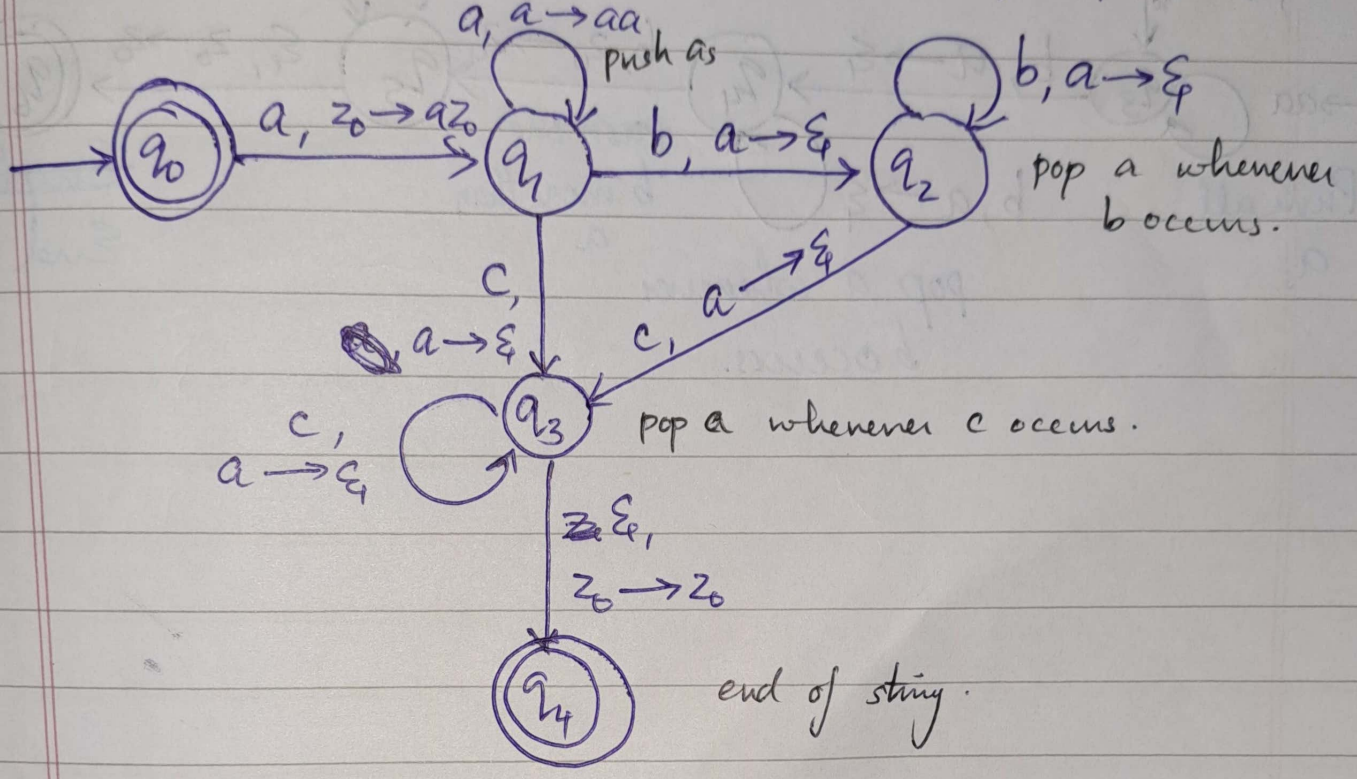
3.



4.



5. $L_5 = \{\varepsilon, aabc, aaabbc, aaabcc, ac, ab, \ldots\}$



2/12/24/Monday.

Type 2 Grammar - Context Free Grammar:

Format:

$A \longrightarrow \alpha$

α - can be terminal / variable / any combination.

Restriction - the left side can only have a single variable.

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

(production rule.)

$S \longrightarrow aSb \mid \epsilon$

CFG

↓ generates

C.F.L

↓ accepted by

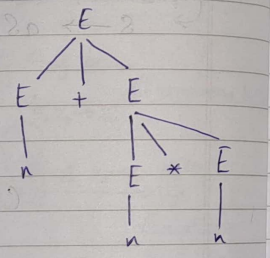
PDA

Applications of CFG:

~~* Specify syntax rules~~
1. Specify syntax rules.
   1. ~~Function~~ / Method structure
   2. Syntax for control structure
   3. Arithmetic expressions' syntax
   4. Checking the balancing of brackets

eg: $E \rightarrow E+E \mid E*E \mid n$, $n \in N$

$n+n*n$ — derivative → tree



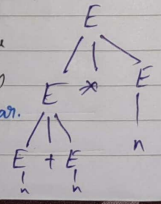
$= n+n*n$

$E \Rightarrow E+E$
$\Rightarrow n+E$
$\Rightarrow n+E*E$
$\Rightarrow n+n*E$
$\Rightarrow n+n*n$

leftmost derivative.

There are two derivation trees for this string. ~~If there~~ Such grammar where there exists more than one derivation trees is called **Ambiguous grammar**. for a ~~string~~ string.

or



CFG $\rightarrow G_1$ —language generated→ $L(G)$

string $\rightarrow w$ whether

How to check, $w \in L(G_1)$?

1. Try to draw the derivation tree.
2. Apply an algorithm to check whether it belongs.
   ↓ CYK Algorithm.
   (coke, Yoeng, Kasami)

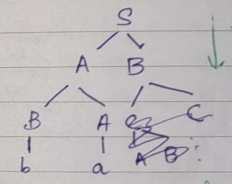
CYK Algorithm: (Membership problem)

To check whether $w \in L(G_1)$.

eg: $S \rightarrow AB \mid BC$
$A \rightarrow BA \mid a$
$B \rightarrow CC \mid b$
$C \rightarrow AB \mid a$

$w = baaba$
check if $w \in L(G_1)$

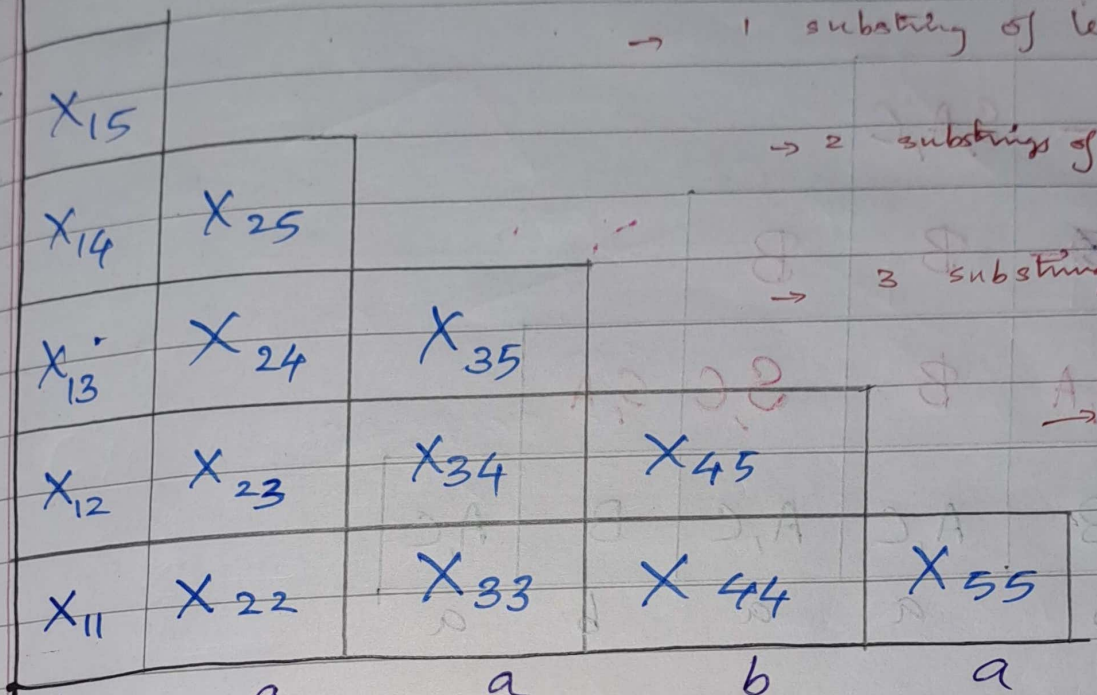
3/12/24 Tue.



↓ top down approach.
↳ This is what we usually follow

But CYK → follows bottom-up approach

Now make table , length = len(string)



X_{11} → 1 to 1 → b
X_{23} → 2 to 3 → aa
X_{25} → 2 to 5 → aaba.
⋮ etc

} substring naming.

Now we see which production gives each of those substrings.

5	S,A,C				
4	—	S,A,C			
3	~~A~~	B	B		
2	S,A	B	S,C	S,A	
1	B	A,C	A,C	B	A,C
	b	a	a	b	a

B → b A → a C → a

X_{12} → ba → B(A,C) → BA, BC

↓ ↓

A S → S, A

X_{23} → aa → ~~A(A,C)~~ (A,C) (A,C)

→ AA, AC, CA, CC

↓ ✗ ✗ ↓

B

X_{24} → ab → (A,C)(B)

AB, AC → S, C

↓ ✗

S,C

X_{45} → ba → S, A

3rd level, (len = 3)

X_{13} → baa → ~~B(A,C)(A,C)~~

↳ divide into two parts

b, aa or ba, a

↓ ↓ ↓ ↓

B B (S,A) (A,C)

(from len=2)

↓

SA, SC, AA, AC

✗ ✗ ✗ ✗

Thus no grammar available.

aab

aa, b

B B

a, ab

(A,C) (S,C) → AS, AC, CS, CC

→ B

B

abaa

ab, a a, ba

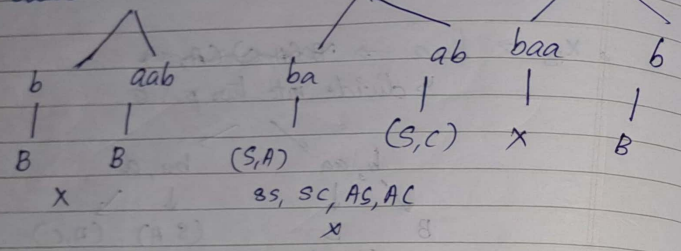
(S,C) (A,C) (A,C) (S,A)

SA, SC, CA, CC AS, AA, CS, CA

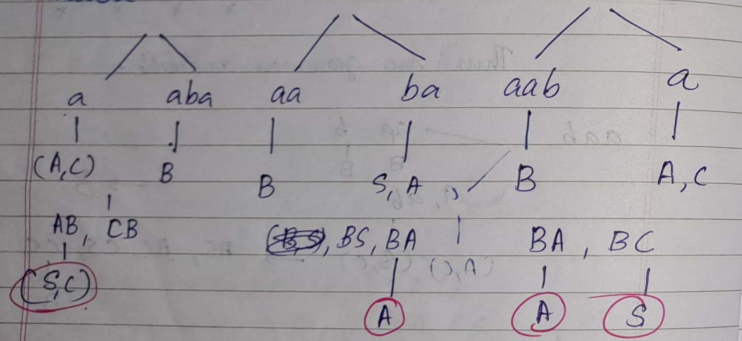
B

Length = 4.

* baab:

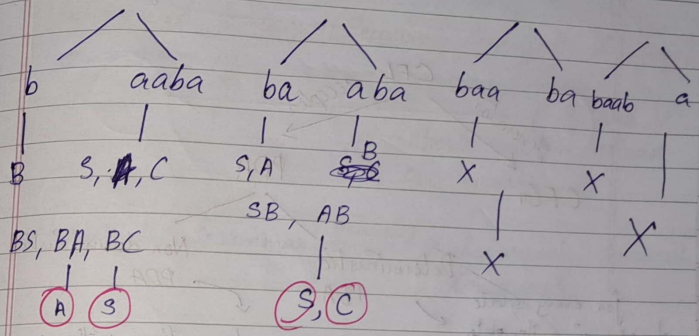


* aaba



Level 4:

baaba.



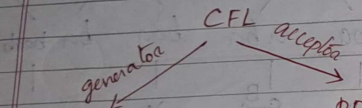
In whichever block S is present, those words ~~can~~ be are present in the language.
That is;
baaba, aaba, ba, ab all are present within the language.

baaba is our word to be checked if it is present in the language.

Lab Assignment – CYK Algorithm.

Continued from 19/11/24. (The qn left unanswered)

$L_2 = \{ ww^R \mid w \in \{0,1\}^* \}$



Deterministic PDA

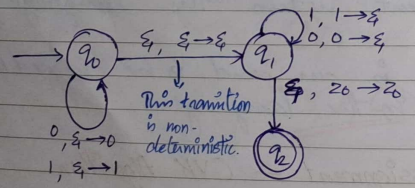
for every state for the given i/p ~~state~~ symbol and stack top there will be atmost one transition. (ε transitions allowed)

Non deterministic PDA

More transitions with same input & stack top. (ε transitions allowed)

Q. $L = \{ ww^R \mid w = \{0,1\}^* \}$

A:



Logic: pushing until half of the string & then popping them.

5/12/24/Thu.

* we are skipping Type 1 Grammar - context sensitive grammar - and directly going to Type 0 grammar.

Unrestricted Grammar (Recursively Enumerable Language)

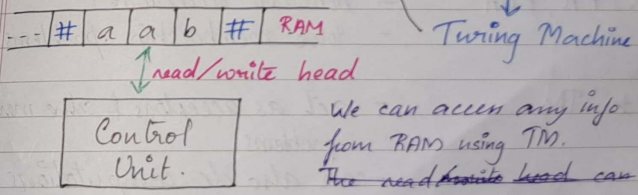
Type 0 grammar.

$\alpha \rightarrow \beta$

α, β - string of T and V

α - should contain atleast one variable.

Unrestricted Grammar → generates → Recursively Enumerable Language.



We can access any info from RAM using TM.

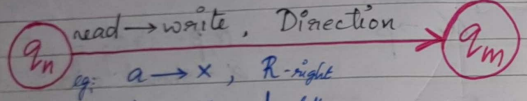
~~The read/write head can move in both directions~~

The read/write head can move in both directions.

We assume all other memory other than the memory we use to save our string as blank. These blank

spaces are represented by '#'

How to mark the transitions in TM?



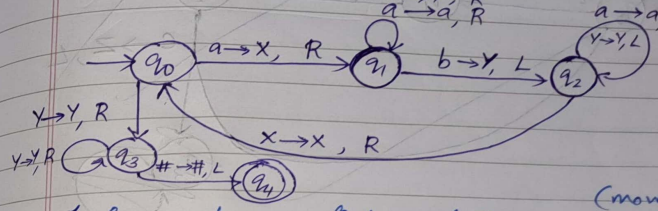
$$\delta(q_n, a) = (q_m, X, R)$$

★ FA, PDA
- both are acceptors
- solve membership problems.

★ TM
- act as acceptors & solve membership problems
- can also do computations ~~unlike~~
(unlike FA & PDA.
↳ called as transducers.

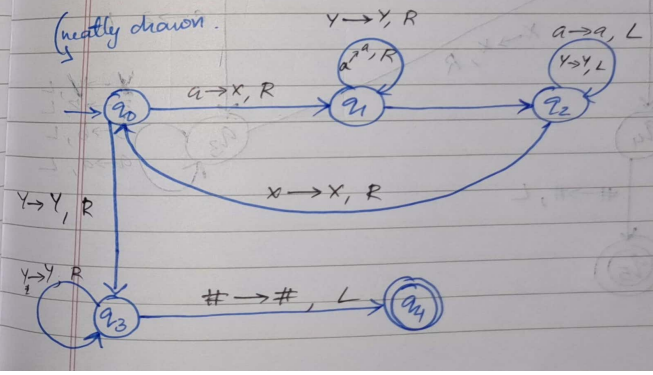
eg: $L = \{a^n b^n \mid n \geq 1\}$ $w = aaa\ bbc$

Logic: each time a comes make it as x each to encode into y.



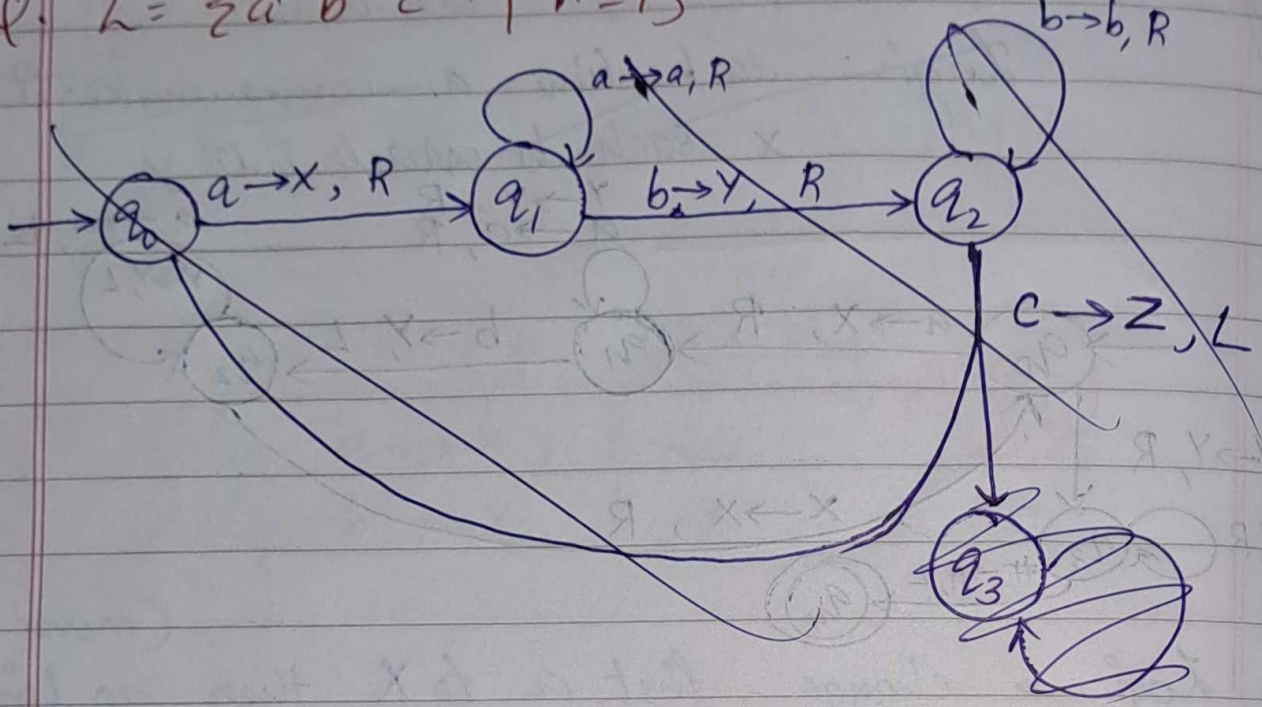
Logic → change first a to X then go find b. (move R?)
Make b to Y. Go left & find x.
Go to right of x find a. Repeat until you get Y after X.
Now skip all Ys and reach blank then end.

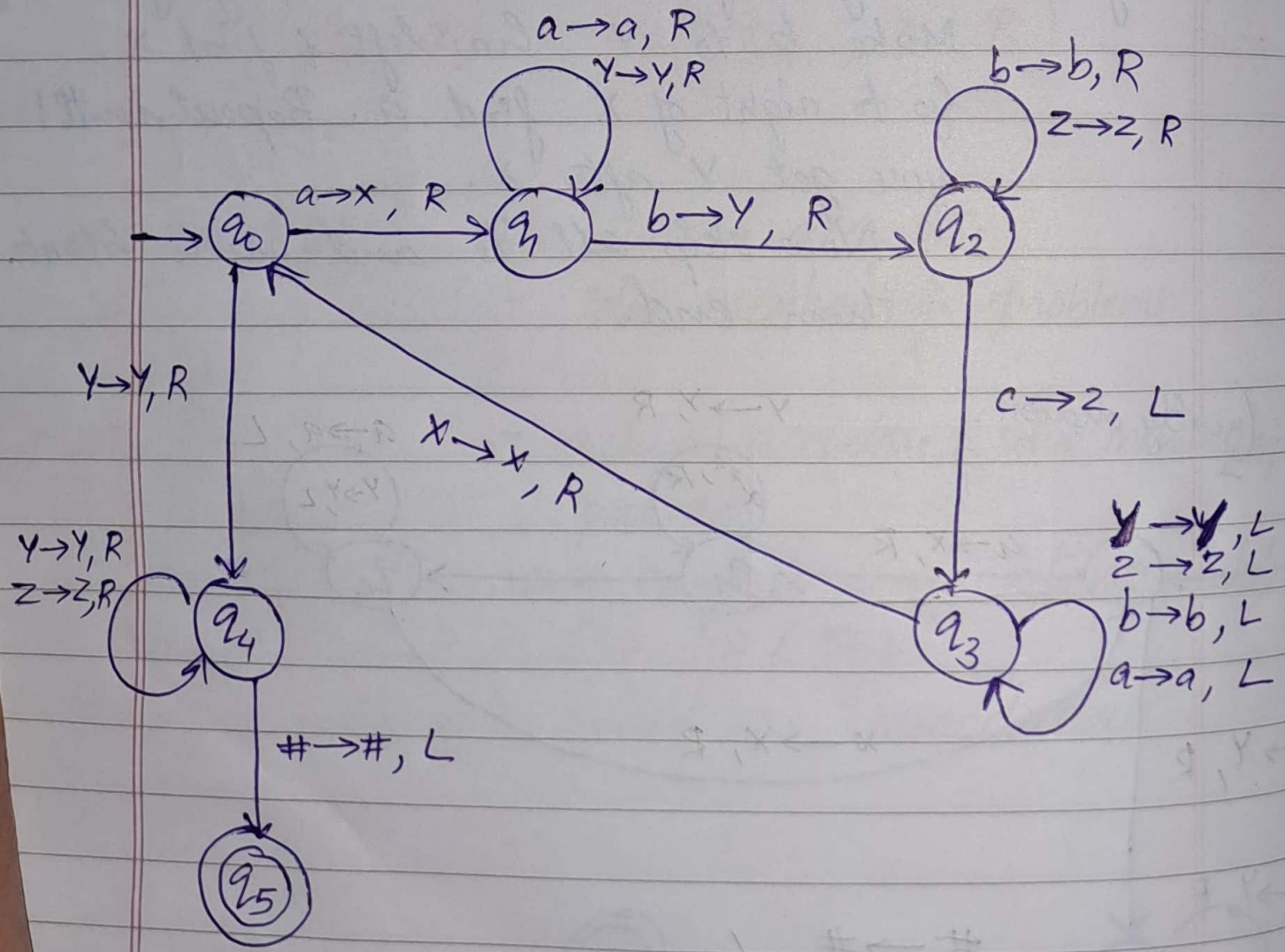
(neatly drawn.



Q. $L = \{a^n b^n c^n \mid n \geq 1\}$

A.



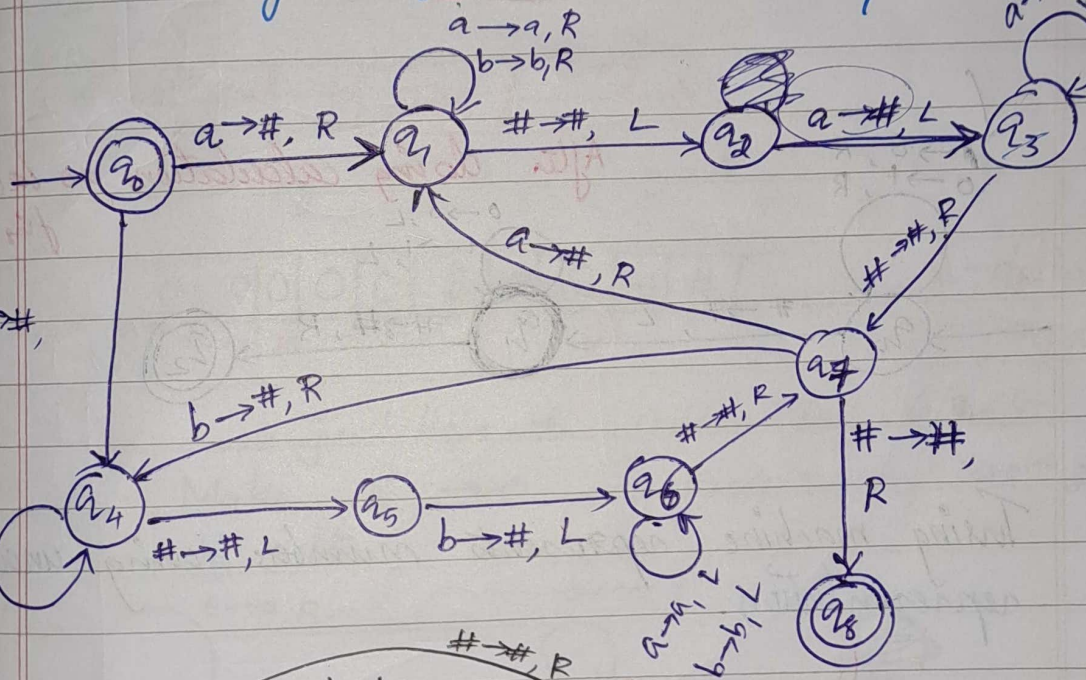


9/12/24/ Mon.

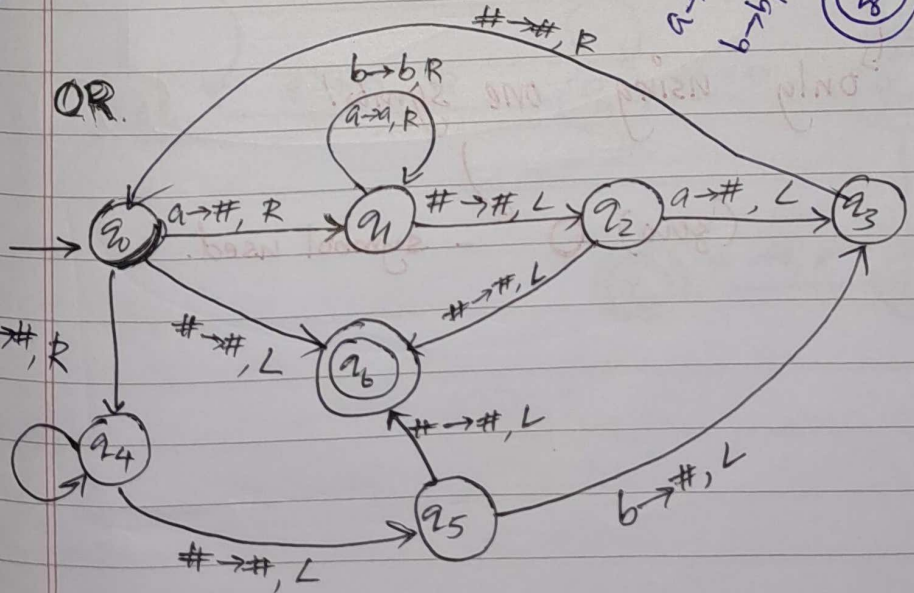
~~Q. L = $\{a^n b^n c^n \mid n>0\}$~~

Q. Design a TM to accept palindrome strings over $\Sigma = \{a, b\}$.

A: Logic: Read first letter - make it blank & go check last letter. Repeat.

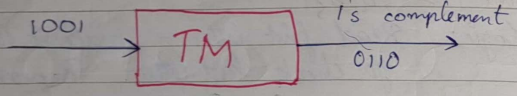


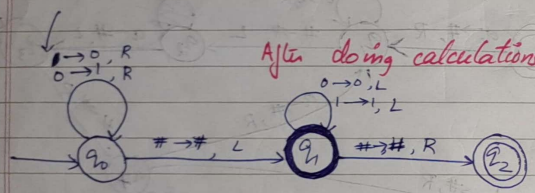
OR



10/2/24/Tuesday.

Turing Machine as a Transducer:





After doing calculations go back to just.

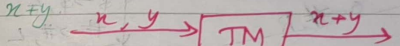
Turing machine represents numbers using unary representation.

↳ only using one symbol

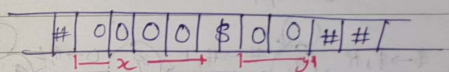
(zero) O - symbol used.

eg: 4 → 0000
3 → 000
2 → 00
⋮ etc

How to do ~~transc~~ operations?



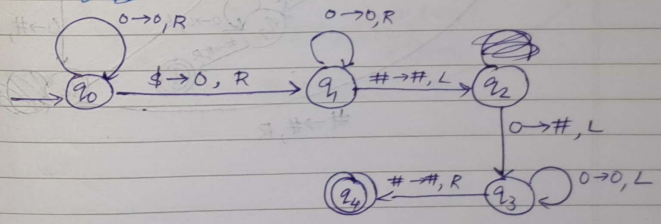
eg: x = 4 y = 2.

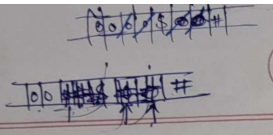


$-delimiter/separator.

Thus we just have to remove that delimiter.

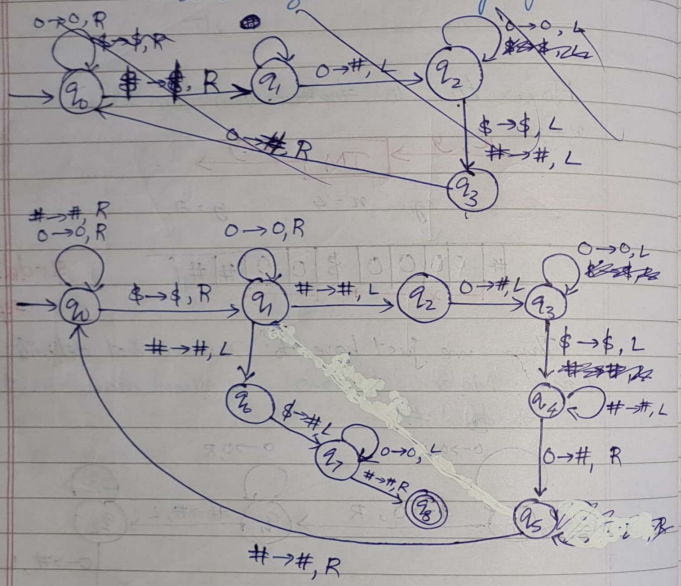
Logic: Make $ → 0 & then make last zero to ~~#~~ #.





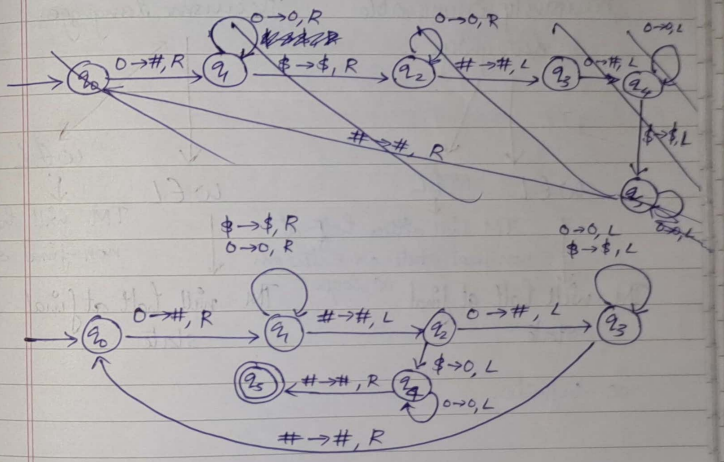
x-y

Logic: For every zero to the right of $ make a zero to the left of $ to #

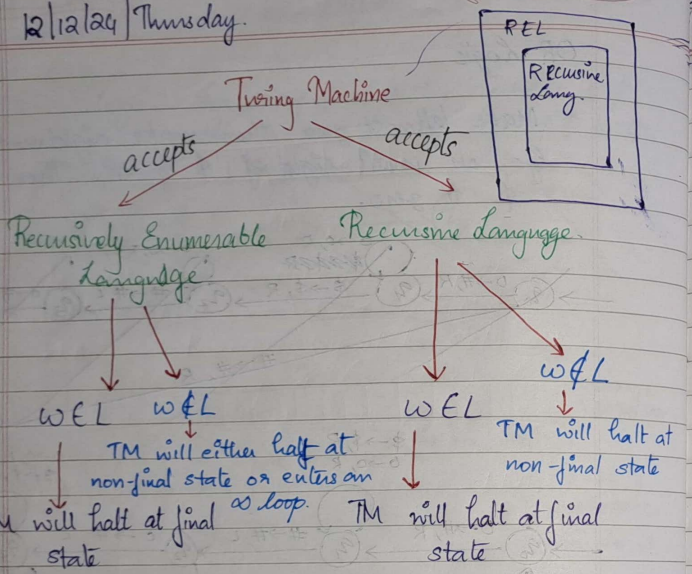


OR Logic

Make leftmost zero then make right most zero. Go on until right of $ is #. Then make $ to zero.



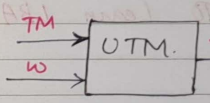
12/12/24 | Thursday.



Universal Turing Machine

- We generally define a TM for a particular language.
- Universal TM is a concept. It is a TM which can be used for anything.
- Can we design a general TM which can tell us whether a particular word can be accepted by a machine.

Halting Problem (The concept behind universal TM)
Can we design an algorithm that can decide whether a given TM accepts or rejects the given string?



In reality UTM is not possible.

Context Sensitive Grammar

- Type 1 Grammar.
- We have to see the context; ie; the surrounding elements.

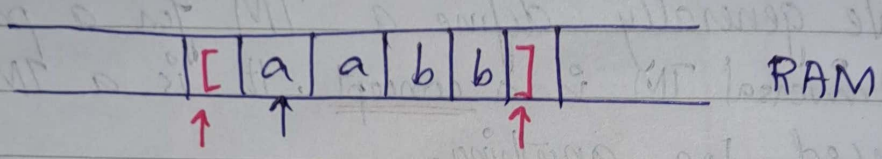
$$\alpha A \beta \longrightarrow \alpha B \beta$$

(context: α, context: β)

where $\alpha, \beta \in (V \cup T)^*$, $B \in V^*$, $B \neq \varepsilon$

is a restricted TM.

Machine accepting CSG → Linear Bounded Automata



In TM we can access any part of the RAM but in LBA we have restricted access.
LBA cannot access beyond the delimiters.
Say length of string = 4 (as above) the machine access is restricted to those 4 areas.

We DO NOT HAVE TO Learn LBA.