User maskable VS unmaskable

- Exceptions can be maskable or unmaskable they can be masked or unmasked by a user task. This decides whether the hardware responds to the exception or not.

We may have instructions that enable or disable exceptions.

~~Within VS Between instructions~~

- ~~Exceptions may have to be handled~~

Theory of Computation

LAG → Language Automata Grammar

→ ↳ Smallest Unit (Symbol)

Alphabet → Σ (finite set of Symbol).

Σ (a, b)

String is a sequence of alphabets.

ex, Σ (a, b) ↓

a, b, aa, ab, ba, bb

If Length = 2, then, aa, bb, ab, ba.

and so on.

(ex) L_1 = Strings of length 3 from Σ (a, b)

= 8

L_2 = Strings start with 'a' end with 'a'.

Automata

Language

↙ Finite ↘ Infinite → L_2

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

L_1 = length 2

{aa, ab, ba, bb}

L_2 = at least a.

$L_1 = \{aa, ab, ba, bb\}$

$= \{abba\}$ [Check every string is present in the main set].

→ For infinite no. of strings, we use the concept of Automata to check whether every string is present in the set of infinite set or not.

Basically Language is a part of String or not is checked by Automata.

F.A PDA LBA TM.

Power of Σ

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

Σ^0 → min. Power of Sigma
= Set of all strings with length '0'.
= λ or ε (NULL String)

Σ^1 = " " " with length 1.
ex: $\Sigma(a)$

Σ^2 = " " length 2.
ex: $\Sigma \cdot \Sigma \{a, b\}\{a, b\}$
$= \{aa, ba, ab, bb\}$
Finite language. Length = 2.

⋮

Σ^* (Kleene Closure) — Set of all strings taking a, b of all length possible.
↳ Infinite Language.

---

here, Possible no. of strings = 2^n

$\Sigma^* - \varepsilon = \Sigma^+$ (⊗ Positive closure)

Grammar: A grammar 'G' is defined as quadruple

$G = \{V, T, P, S\}$ → Start
- V → Variable
- T → Terminal
- P → Production Rule

$S \to aSb \mid \varepsilon \to$ NULL

S → Start symbol

aSb → Terminal

Put $S = aSb$

$= \varepsilon, aSb, aaSbb$

$= \varepsilon, ab, a^2b^2, a^3b^3 \ldots a^nb^n$ $n \geq 0$

$n=0 \Rightarrow \varepsilon$

$n=1 \Rightarrow ab$ and so-on.

Q-1) Can we generate abab using $S \to aSb \mid \varepsilon$

↳ No we cannot.

$S \to aSb \mid \varepsilon$

$= aaSbb$, $a\varepsilon b$

$= a^2b^2, ab$

So, ~~aaSbbε~~ abab is never generated

Q

$S \to SS$

$S \to aSb$

$S \to bSa$

$S \to \varepsilon$

ε, aSb, bSa

↓

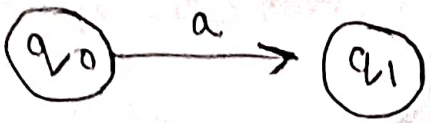
$\varepsilon, aaSbb, bbSaa$

$\varepsilon, a^2b^2, b^2a^2$

This generates a language, where $n_a(\omega) = n_b(\omega)$.

DFA (Deterministic Finite Automata)

↳ DFA $(Q, \Sigma, \delta, q_0, F)$



→ Transition

$Q, \Sigma, \delta, q_0, F$ → Set of finite final states

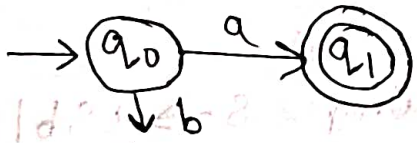
- Q → Set of finite States
- Σ → Set of alphabets
- q_0 → Initial State
- F → Set of finite final states

Whenever an alphabet is applied to an state, it goes to a ~~state~~ state and this is deterministic. (goes to single State)

$F \subseteq Q$ $\quad \Sigma(a,b)$

Q → Strings starting with a

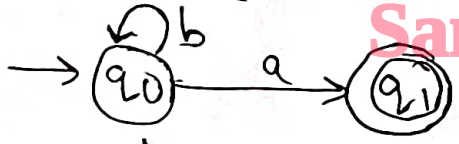
$\{a, aa, aaa, ab, \ldots\}$ → infinite



['b' cannot reach q_1]

Final State.

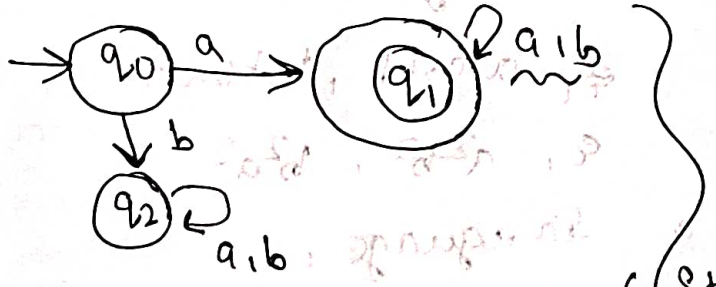
∴ trap 'b'



→ X

because, this represents ba.

∴ dead State, always variables a, b are included and trapped.



Final DFA

6 States

$\delta: Q \times \Sigma \to Q$

$q_0 \times (a,b) =$

q_1

q_2

$q_0 a$ → q_1

$q_0 b$ → q_1

$q_1 a$ → q_1

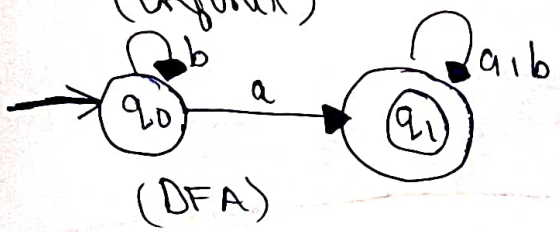
$q_1 b$ → q_2

$q_2 a$ → q_2

$q_2 b$ → q_2

Construct a DFA which accepts a language of all strings.

↳ end with 'a'
↳ containing 'a'
↓
{a, aa, aaa, b . . .}
(infinite)



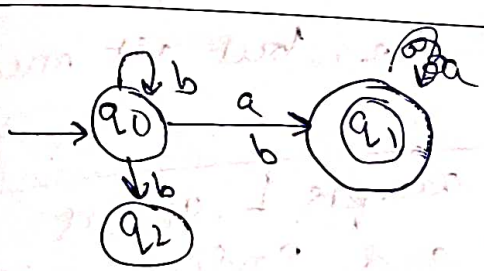
(DFA)

↳ It accepts all strings which contains 'a'.

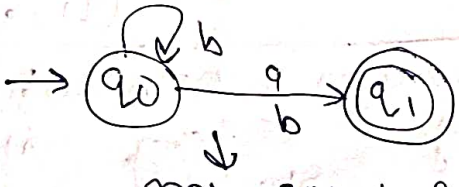
every test case is handled.

* We draw self-loop, for any variable a, b, c etc, when, there occurrences of these variables are more than once.
ex) aabbc aa etc.

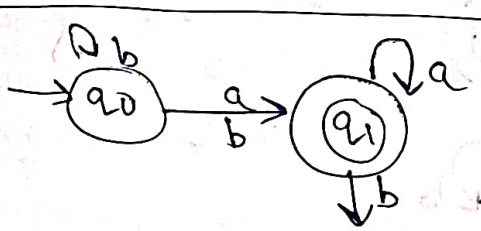
ex)



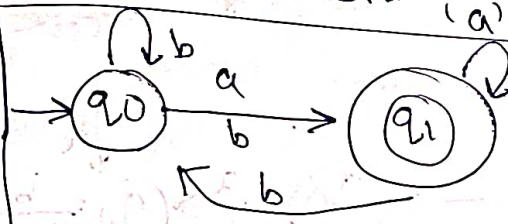
we cannot design trapstate, because then 'ba' won't be accepted



not accepted because, if we give 'b' it will be accepted but it doesn't contains 'a'.



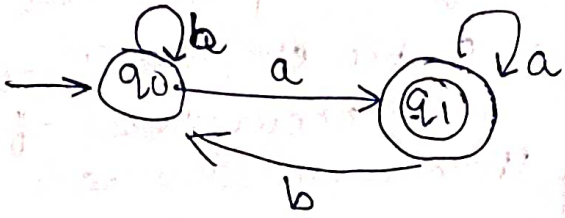
is not possible, because, then string aaba won't be accepted. we must return to the final state (q1) and for above DFA, this won't happen.

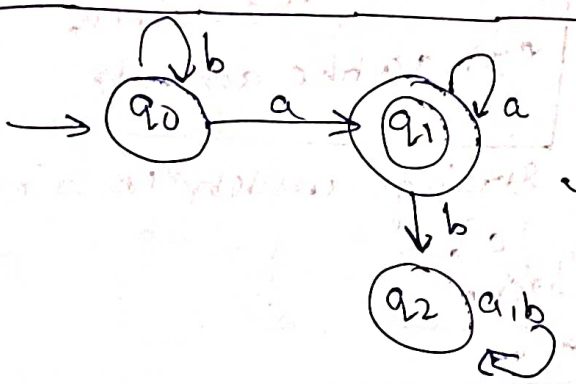


DFA is not possible, because aab, will not be accepted, as it will return to q0, but it must return to (q1), so this DFA isn't accepted.

→ ending with 'a'

$L = \{ a, aa, aaa, ba \ldots \}$

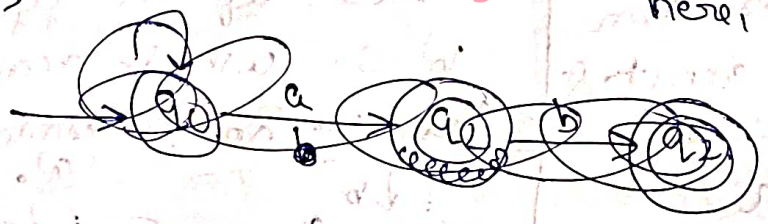


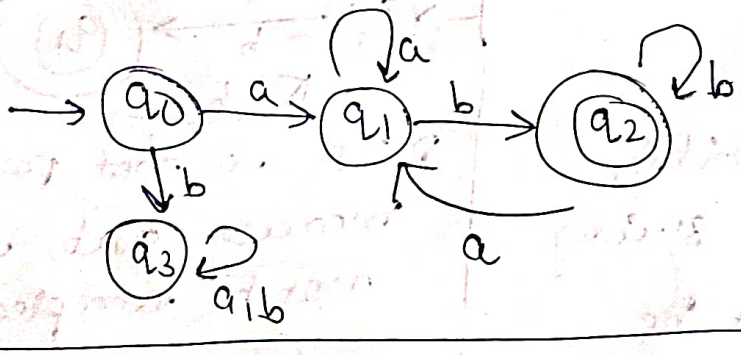


is not possible, bcoz, aba is not accepted, as this ends at q_2, but it must end with at q_1.

Q Construct a DFA which accepts L, of all strings ~~Starting~~ with 'a' and ending with 'b'.

here, $L = \{ ab, abab, \ldots \}$





Q not Starting with 'a' or not ending with 'b'

A B

here, $L = \{ \lambda, \varepsilon, a, b, ba \ldots \}$ etc

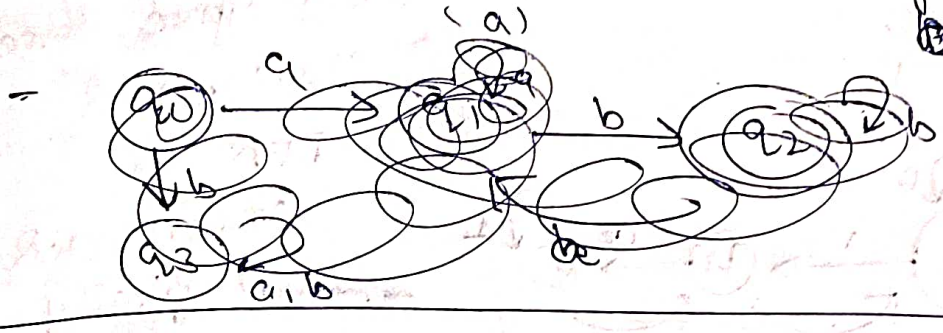
$(A \cup B)' = A' \cap B'$

A = not starting with a (let)
B = not ending with b.

So, $(A \cup B)^C$ → not

$= A^C \cap B^C$

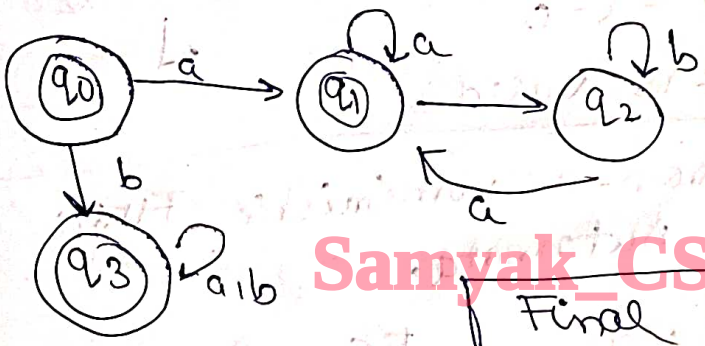
OR = Starting with a and ending with b.



---

here, final → not - final

non-final → final



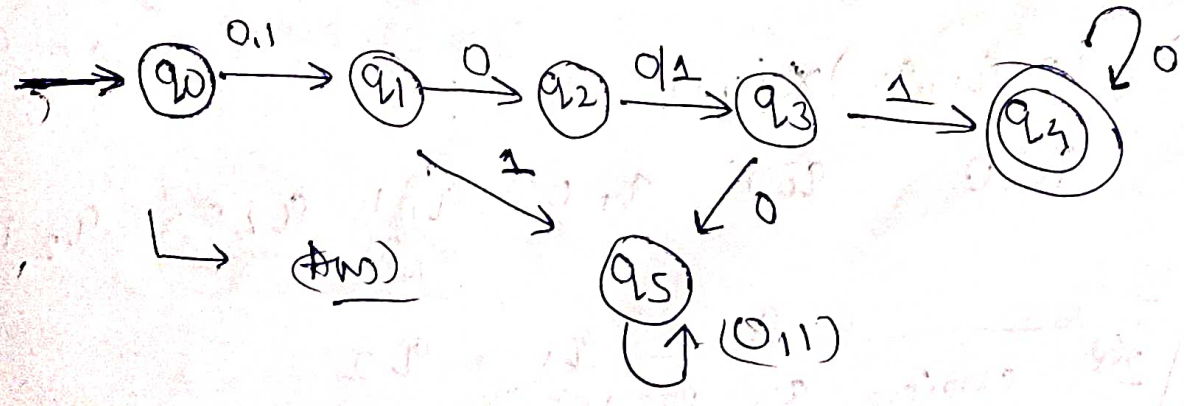
Final DFA

Q Design a DFA, which accepts all strings over (0,1) in which second symbol is '0' and fourth symbol '1'.

Soln. 0/1 0 0/1 1 length = 4

min. states in DFA = (n+1).

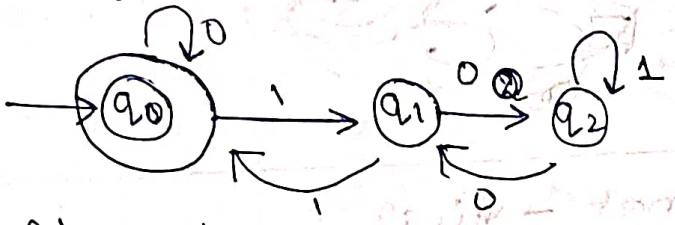
here n+1 = 5



(Q) construct a DFA which accepts a language of all binary strings divisible by three over $\Sigma(0,1)$

→ Remainders possible are 0,1,2. → now, these remainders are assigned some states, and DFA is constructed finally.

$q_0 - 0$
$q_1 - 1$
$q_2 - 2$



Check for any no. : $(12)_{10} = (1100)_2$

$12 \div 3 = 0$

So check, it is valid.

NDFA | NFA → Non-deterministic Finite Automata.

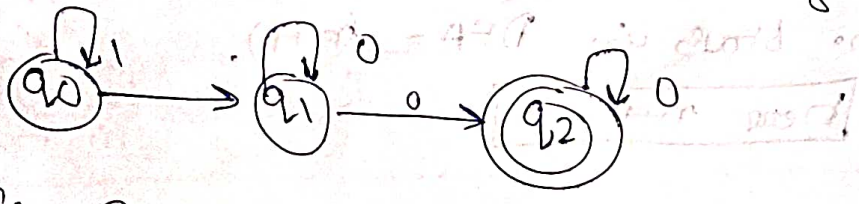
↳ $(Q, \Sigma, q_0, F, \delta)$, $F \subseteq Q$

Final State — Set of all finite states.

→ Here choices of moves are present.

→ NFA is used for regular languages.

ex)



$\delta: Q \times \Sigma = 2^Q$

ex) $(q_0, q_1, q_2) \times (0,1)$

$= q_0,0 \quad q_0,1 \quad q_1,0 \quad q_1,1 \quad q_2,0 \quad q_2,1$

here 2^Q choices are present

	q_0	q_1	q_2

DFA	NFA
1) Dead - configuration is not allowed.	1) Dead configuration is allowed.
2) Multiple choices aren't available here.	2) Multiple choices are available.
3) ε- move is not allowed.	3) ε- Move is allowed.
4) Digital computers are deterministic.	4) Non-deterministic feature is not associated with real computers.
5) Designing/understanding is difficult.	5) Designing/understanding is easy.

* $q_0 \xrightarrow{0} q_1$

here, we aren't defining where 1 is going, this is dead configuration and this isn't allowed in DFA.

* $q_0 \xrightarrow{\varepsilon} q_1$ → This is ε- Move.

(Q) NFA of all binary strings in which 2nd last bit is 1.

(Ans) →

$L = \{ \underline{\quad 1 \quad} \underline{0/1} \ldots \}$ → ex) 10, 11, 010, 110, 111, 1111 0.

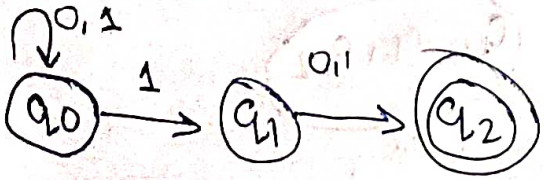
$(0+1)^* 1 (0+1)$

↓ (0 or 1) ↓ (Fixed) ↓ (0 or 1)

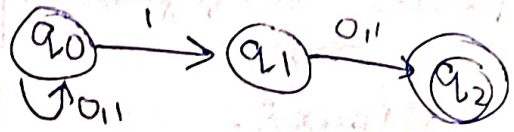
Max states = (n+1)

= 2+1

= 3

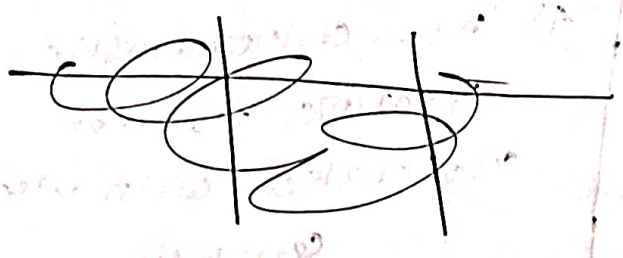


(Q) NFA of all binary strings in which 2nd last bit is 1.



	0	1
q_0	q_0	$q_0\ q_1$
q_1	q_2	q_2
(q_2)	—	—

→ Transition Table



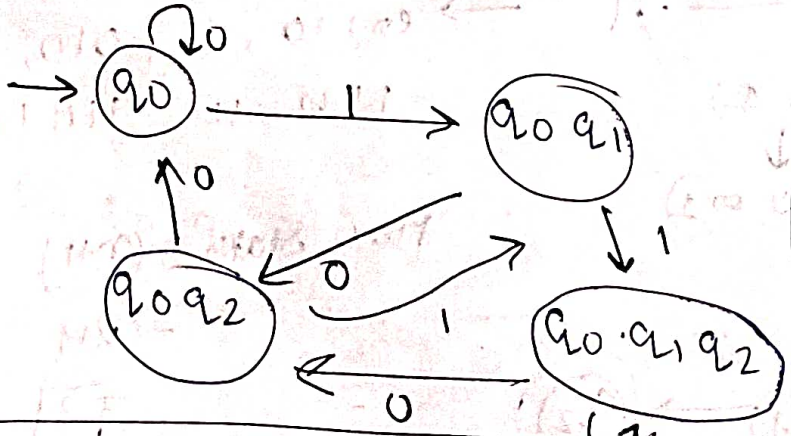
↓ check NFA table to draw DFA table expand every term.

	0	1
q_0	q_0	$q_0\ q_1$
$q_0\ q_1$	$q_0\ q_2$	$q_0\ q_1\ q_2$
$q_0\ q_2$	q_0	$q_0\ q_1$
$q_0\ q_1\ q_2$	$q_0\ q_2$	$q_0\ q_1\ q_2$

(First row, as it is copied)

→ We see that all states have been expanded.

now, draw each of the states



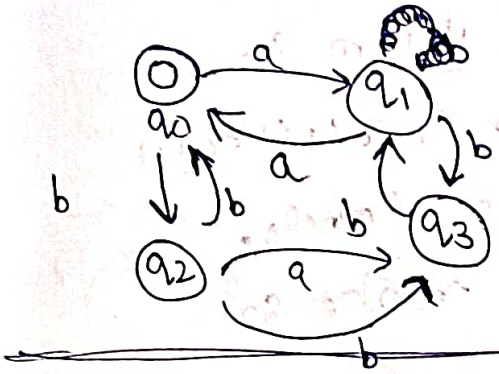
DFA Diagram

Final State was q_2 (NFA), here also final state are those states, where q_2 appears.

(a) DFA for even a and even b. / even a odd b / odd a and even b / odd a odd b.

(a) let $L = \{ w \mid w$ has even no. of a's and even no. of b's $\}$

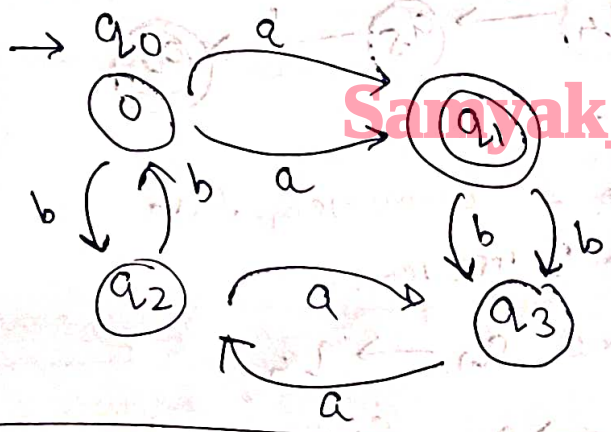
Soln → $L = \{ \varepsilon, aa, bb, abab \ldots \}$



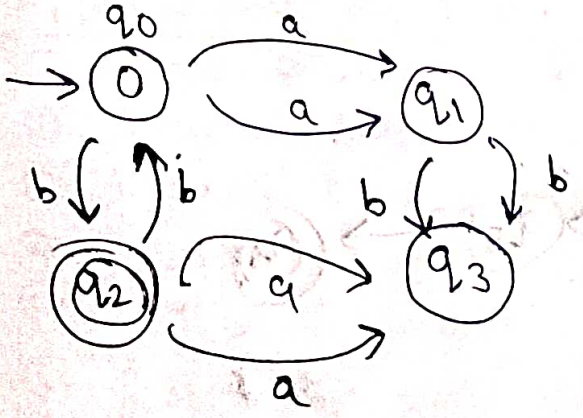
← DFA

When (0) was made the final State.

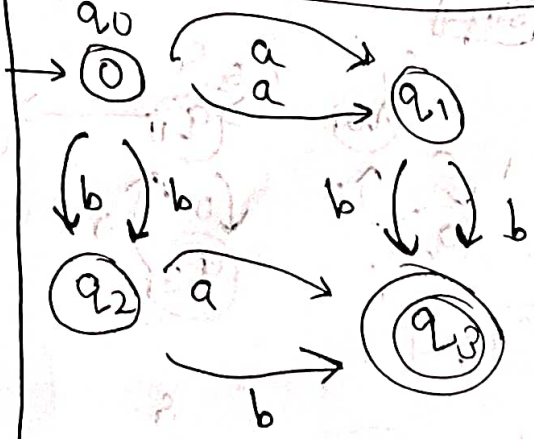
Make, q_1 the final State, if you want odd no. of a and even no. of b.



q_2 = even a and odd b

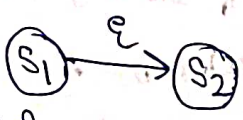


q_3 = odd a and odd b.

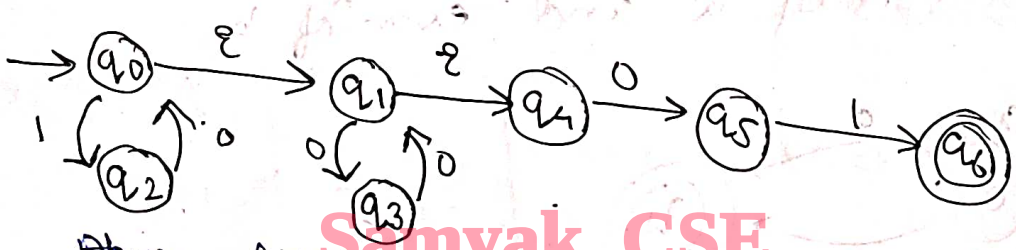


Eliminate epsilon ε-moves | Conversion from epsilon nfa to nfa

ε-NFA (epsilon NFA) → eliminating ε-Moves.



1) Find all edges starting from S_2.
2) Duplicate all edges to S_1, without changing edge labels.
3) If S_1 is initial state, make S_2 initial.
4) If S_2 is final state, make S_1 final.

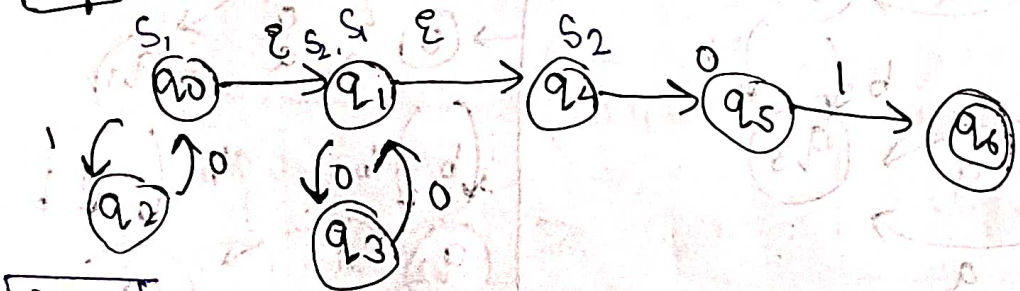


Above diagram has 2 ε-moves, let's eliminate these 2 moves.

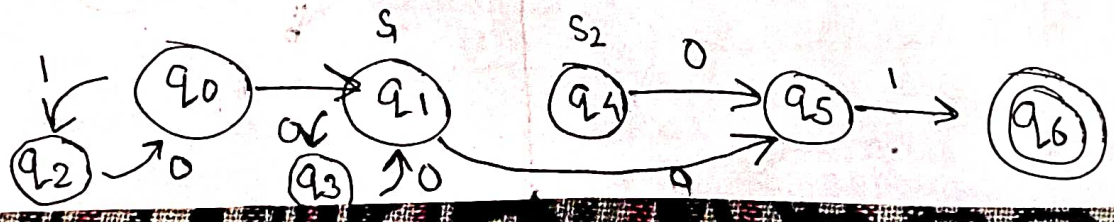
i.e., $q_0 \rightarrow q_1$ and $q_1 \rightarrow q_4$

→ First eliminate $q_1 \rightarrow q_4$ ε-move bcoz, after q_4, we don't have any such ε-move.

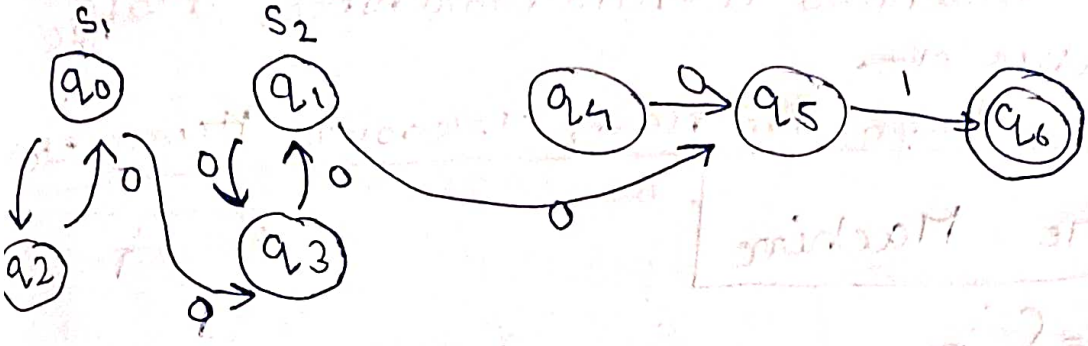
Step-1



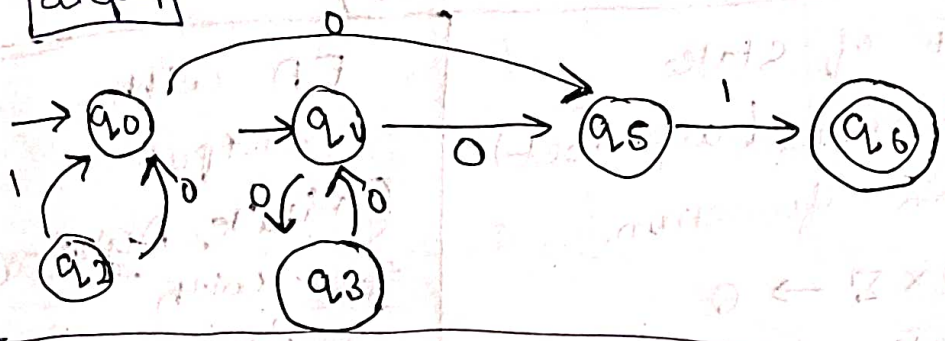
Step-2



Step-3



Step-4



* q_4 was dead state (means, it wasn't connected to any initial state) thus it is removed.
* here, since, S_1, was initial, so, S_2 is also made initial.
* also, here, as S_2 was not final, so S_1 was also not made final.

Limitations of FSA

1. Limited memory
2. Strings without comparison
3. Linear Power.

Applications of FSA

1. word processor program
2. Digital logic design.
3. Lexical Analyzer
4. Switching Circuit Design.
5. Text Editors etc.
6. Software for scanning large bodies of text such as web pages to find occurrence of words, phrases, Patterns.

* Software with finite states like vending machines, weight machines, traffic lights etc.
* Game design (Pac Man, Treasure Hunt etc).

Moore Machine

$M_0 = \{ Q, \Sigma, \Delta, \delta, \lambda, q_0 \}$

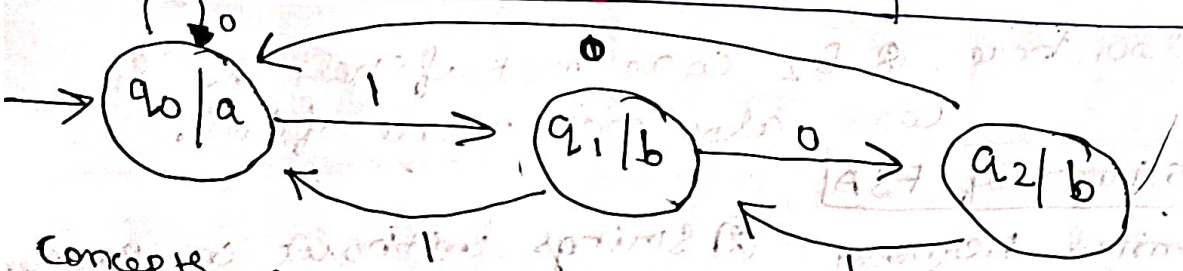
Q = finite set of states
Σ = input symbol (alphabet)
δ = Transition function.

$$Q \times \Sigma \rightarrow Q$$

q_0 = Start State
Δ = output symbol
λ = Output function

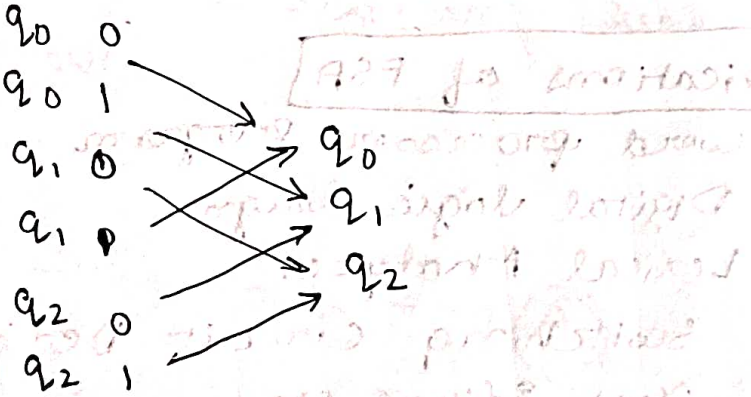
$$\lambda : Q \rightarrow \Delta$$

FA with output (Finite Automata with output)



Concepts are same like DFA.

q_0
q_1 x (0|1)
q_2



Here: $\Delta = (a, b)$

$$\lambda : \begin{cases} q_0 \rightarrow a \\ q_1 \rightarrow b \\ q_2 \rightarrow b \end{cases}$$

For (n) size of input, we get (n+1) size as output.

(ex) 00110, → at first, we arrive at q_0 only, for ε, input.
aaabaa

So, output = aaabaa.

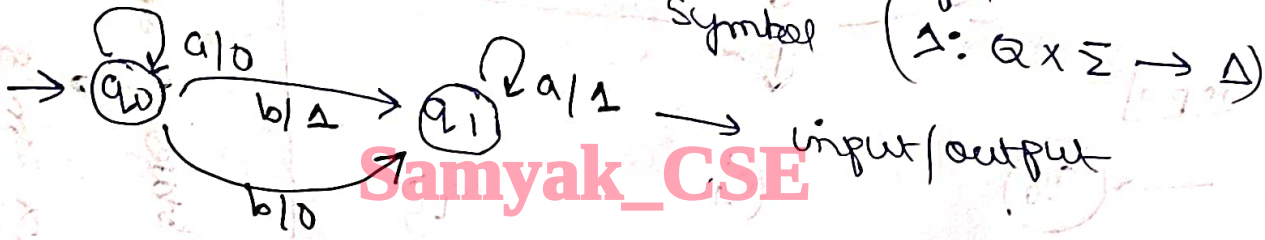
Current State	Next State		Output
	0	1	
q_0	q_0	q_1	a
q_1	q_2	q_0	b
q_2	q_0	q_1	b

Mealy Machine → transition fn, $\delta: Q \times \Sigma \to Q$

↓ FA

$M_e = \{Q, \Sigma, \Delta, \delta, \lambda, q_0\}$

- Q → finite set of states
- Σ → input alphabets
- Δ → output symbol
- λ → output fn ($\lambda: Q \times \Sigma \to \Delta$)
- q_0 → initial state



→ input/output

here a/0 means input/output

δ: ↓ $q_0 \times (a, b)$ ←

q_0 a → q_0

q_0 b → q_1

q_1 a → q_1

q_1 b → q_1

→ It is different from Moore, Machine in the sense, in Moore, $\lambda: Q \to \Delta$

here, $(\lambda : Q \times \Sigma \to \Delta)$

meaning state gives, output which is input dependent.

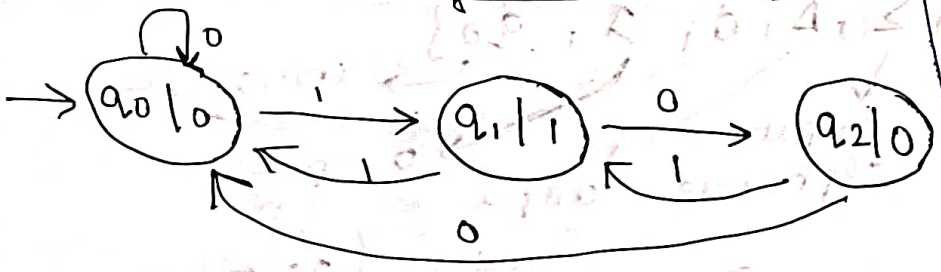
$q_0, a \to 0$ $q_0, b \to 1$ $q_1, a \to 1$ $q_1, b \to 0$

For 'n' size input, output = 'n' size.

Current State	Next State			
	input = a		input = b	
	State	output	State	output
q_0	q_0	0	q_1	1
q_1	q_1	1	q_0	0

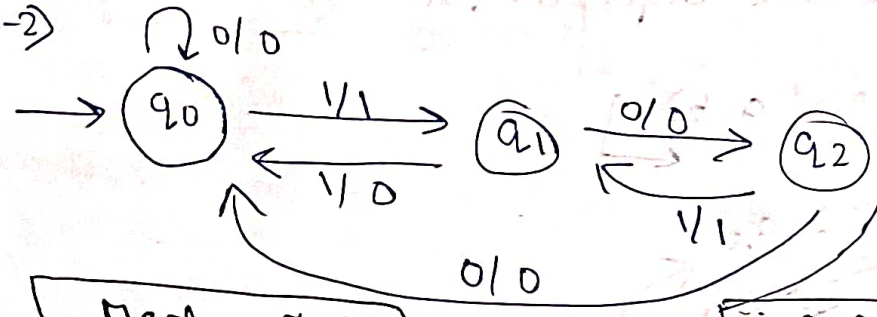
* Mealy machine is faster as, its output is asynchronous.
whereas for moore, machine, output is synchronous with clock.

Moore to Mealy Conversion



Step-1

S-2)



Mealy Table

Current State.	0	1
q_0	q_0,0	q_1,1
q_1	q_2,0	q_0,0
q_2	q_0,0	q_1,1

Write the output just beside to States corresponding

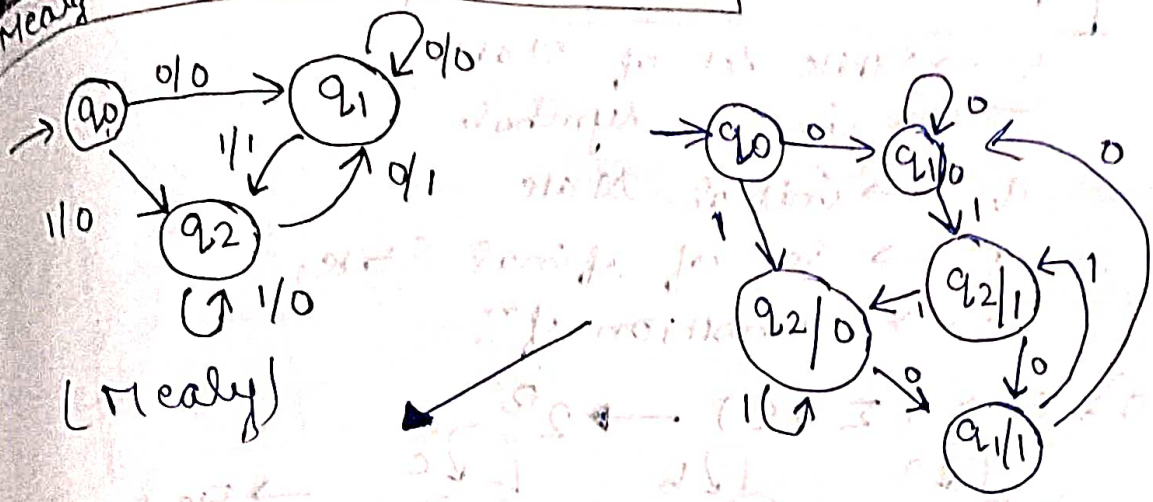
TABLE-1

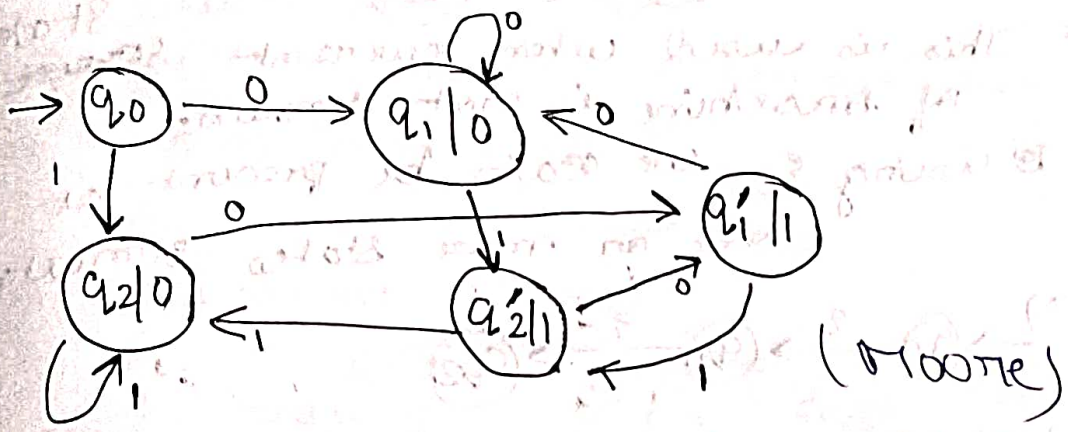
C-S	0	1	Output
q_0	q_0	q_1	0
q_1	q_2	q_0	1
q_2	q_0	q_1	0

(Moore transition Table

take help of Table 1

Mealy to Moore Conversion





For 'm' state, n output in mealy

3 → 2

$(m \times n)$ max. no. of states in moore machine.

Ex: We start with initial (q_0) state.

We make states for each of the inputs (1, 0).

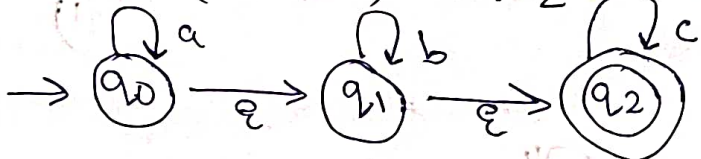
Now move to another state (say q_1) do the same steps.

Further, outputs contradicts for same input, make another state (duplicate of the state you're taking) and do the steps.

Epsilon NFA (ε-NFA)

$Q \rightarrow$ Finite set of states
$\Sigma \rightarrow$ input symbols
$q_0 \rightarrow$ initial state
$F \rightarrow$ set of final states
$\delta \rightarrow$ transition fn.

$\delta : Q \times (\Sigma \cup \varepsilon) \rightarrow 2^Q$

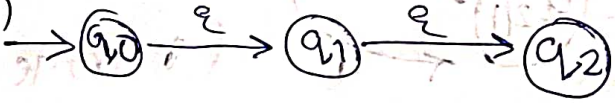


→ Fig-1

↳ This is used when current state of machine is not known.

Using ε, we can be present at one or more states simultaneously.

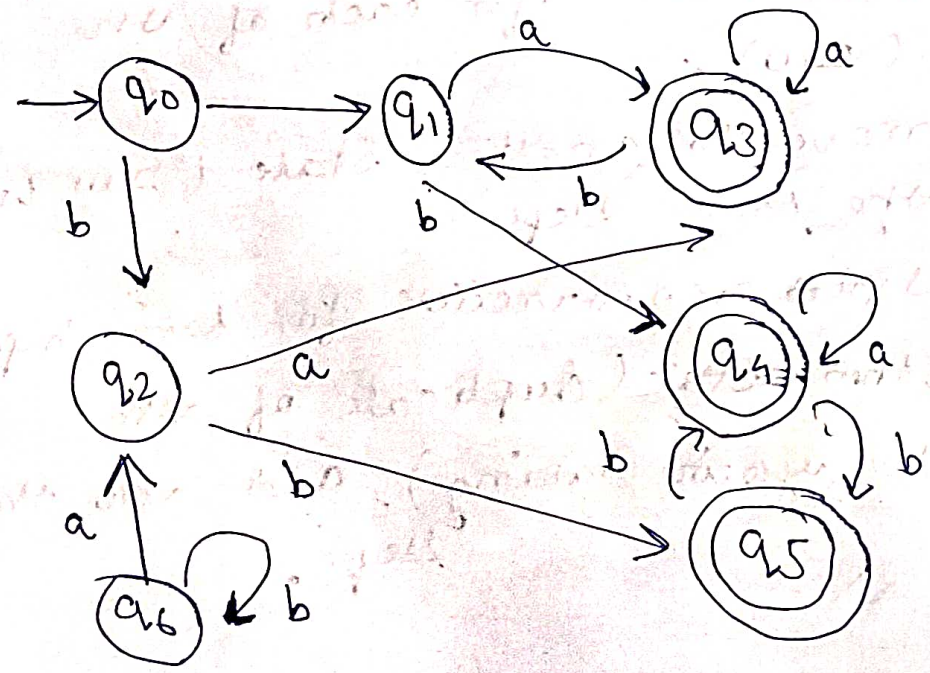
ex)



Using this machine, alone, a, b, c either of 3 symbols can be accepted from Fig-1.

Minimization of DFA

ex)



	a	b
q_0	q_1	q_2
q_1	q_3	q_4
q_2	q_3	q_5
q_3	q_3	q_1
q_4	q_4	q_5
q_5	q_5	q_4
q_6	q_2	q_6

NFA → DFA

For 'n' states → 2^n states (max) can be present.

① Remove unreachable states.
(the states which we cannot reach when starting from q_0).
eg:) q_6

② Make equivalent classes.

Accepting States (Final)

Non-accepting states. (non-final)

zero equivalent class

$\pi_0 = \{q_0, q_1, q_2\}$ $\{q_3, q_4, q_5\}$

(non-accepting) (accepting class)

G_1 (let say) G_2 (let say)

$\pi_1 = \{q_0\}$ $\{q_1, q_2\}$

[we've checked q_0, q_1 For checking, take q_0, q_1 check their transit

	a	b
q_0	q_1	
q_1	q_3	

→ they give q_1 q_3 and q_1 q_3 belong to different gro

so seperate them and write q_0, q_1 in different groups under π_1 where π_1 is a sub-group of π_0 obviously.

now check the other. q_0, q_2 do the same process.

So we obtain π_1 from G_1

$$\pi_1 = \{q_0\} \quad \{q_1, q_2\}$$
$$G_{11} \qquad G_{12}$$

[there is no use of checking q_1, q_2 both, they have already been placed in the same group]

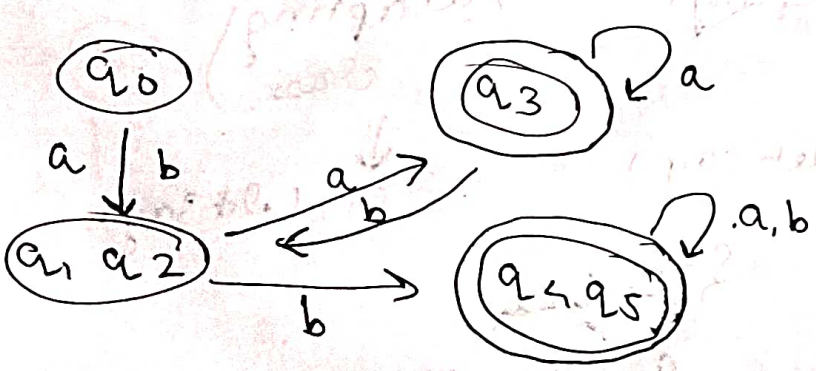
now, form groups for G_2.

So,

$$\pi_1 = \{q_0\} \quad \{q_1, q_2\} \quad \{q_3\} \quad \{q_4, q_5\}$$
$$G_{11} \qquad G_{12} \qquad G_{21} \qquad G_{22}$$

$$\pi_2 = \{q_0\} \quad \{q_1, q_2\} \quad \{q_3\} \quad \{q_4, q_5\}$$

So, we have 4 states left with us, but earlier it was 7.



Minimized DFA

Regular Expressions

↳ Method to represent a language.

→ Any language which is accepted by FA is called regular language.

Using above method, we can represent both NFA and DFA.

Let (R) be a regular expression over alphabet Σ if R is :-

1) λ, ε, is regular expression denoting the set $\{\varepsilon\}$

2) ϕ, is regular expression denoting the empty set $\{\}$

3) For each symbol $a \in \Sigma$, a is a regular expression denoting set $\{a\}$.

4) Union of two RE is also regular.

5) Concatenation of two RE is also regular.

6) Kleene closure * of RE is also regular.

7) If R is regular, (R) is also regular.

8) Nothing else, repeat 1. to 7 recursively

$R = \varepsilon \quad L(R) = \{\varepsilon\}$

$R = \phi \quad L(R) = \{\}$

$R = a \quad L(R) = \{a\}$

} Primitive Language.

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

Kleene Closure * example.

Regular Expression

NOTE
a^* is equivalent to a^n, $n \geq 0$

$+ = * - \varepsilon$

example 1 Using $\Sigma(a,b)$

*Language

↙ Finite ↘ Infinite

1) NO string ex $\{\ \}$ $RE = \phi$
2) Length 0 $\{\varepsilon\}$ $R = \varepsilon, \lambda$
3) " 1 $\{a, b\}$ $R = (a+b)$
4) " 2 $\{aa, bb, ab, ba\}$ $R = (a+b)(a+b)$
5) " 3 $R = (a+b)(a+b)(a+b)$
6) Atmost 1 $\{\varepsilon, a, b\}$ $R = (\varepsilon+a+b)$
7) Atmost 2 $R = (\varepsilon+a+b)(\varepsilon+a+b)$

Regular Expression for Infinite Languages.

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

1) All strings having a single (b)
a^*ba^*

2) All strings having at least one (b).
$(a+b)^*b(a+b)^*$

3) All string having bbbb as sub-string.
$(a+b)^*bbbb(a+b)^*$

Pumping lemma

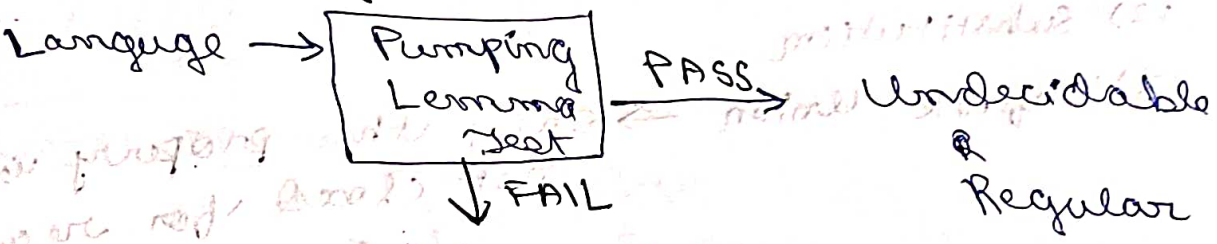
→ If L is an infinite language then there exists some (+ve) integer (n) (Pumping length) such that any string $w \in L$ has length greater than equal to (n), i-e $|w| \geq n$ then w can be divided into three parts, $w = xyz$ satisfy the following conditions:-

(i) for each $i \geq 0$; $xy^iz \in L$

(ii) $|y| > 0$ and

(iii) $|xy| \leq n$

* Pumping lemma is a method to check whether a infinite language is regular or not.
* It is a negative test.



ex) $a^n b^{2n}$ $n \geq 0$.

take $w = aabbbbbb \in L$

$x = aa$, $y = bb$, $z = bbbb$

(Decide x, y, z by yourself).

$w = aa\ bbbb\ bb \notin L$

$y = (bb)^i$ → as $i \to 2$, [Basically we are Pumping y]

$w \notin L$, it is violating cond^n (i)

So, Test fails, hence 'L' is not regular

Closure Properties of Regular Languages

1) Union $(L_1 \cup L_2)$
2) Concatenation $(L_1 \cdot L_2)$
3) Closure (*) L^*
4) Complementation $\bar{L} = \Sigma^* - L$
5) Intersection $L_1 \cap L_2 = \overline{\bar{L_1} \cup \bar{L_2}}$

6) Difference $L_1 - L_2 = L_1 \cap \bar{L}_2$

7) Reversal $(L)^R$

8) Homomorphism

9) Reverse Homomorphism

10) Quotient operation.

11) INIT

12) Substitution

13) Infinite Union → only this property is not closed for regular language.

Regular languages are closed under reversal.

↳

1) Make initial state of M as final state of M'.

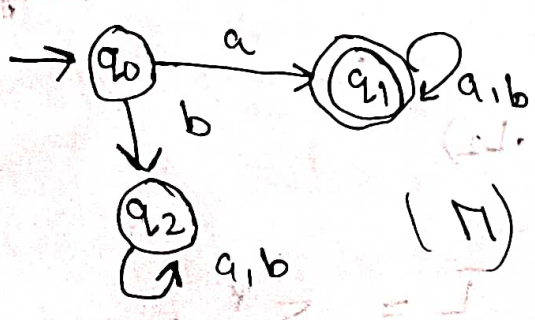
2) Final state of M becomes initial state of M'.

3) Reverse the direction of edges of M to make M'.

4) No change in Loop and remove unnecessary states.

Example: L_1 = { set of all strings over (a,b) starting with a }.

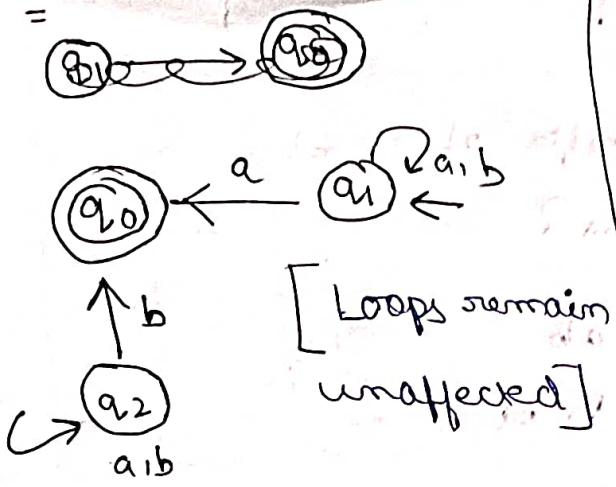
↳ $R = a(a+b)^*$



(M)

$(M)^1$ = Reversal of M

M' =



[Loops remain unaffected]

Quotient operation

→ Left Quotient (cut Prefix)

→ Right Quotient (cut Suffix)

$\frac{L_1}{L_2} = \{x \mid xy \in L_1, \text{ for some } y \in L_2\}$ Right Quotient (cutting in right)

$\frac{L_1}{L_2} = \{y \mid xy \in L_1 \text{ for some } x \in L_2\}$ Left Quotient (cutting in left)

here

→ L_1, L_2 are same languages on, same input symbol Σ.

Let say (0, 1)

ex) $L_1 = \{10, 100, 1010, 101110\}$

$L_2 = \{10\}$

$L_1 = a^*b$; $L_2 = ab$

$\frac{L_1}{L_2}$ (left) = $\frac{10}{10}$, $\frac{100}{10}$,

ε, 0, 10, 1110

$\frac{L_1}{L_2}$ (Right) = ε, ε, 10, 1011, 10

INIT

(initial / Prefix)

{ set of all Prefix of $w \in L$ }

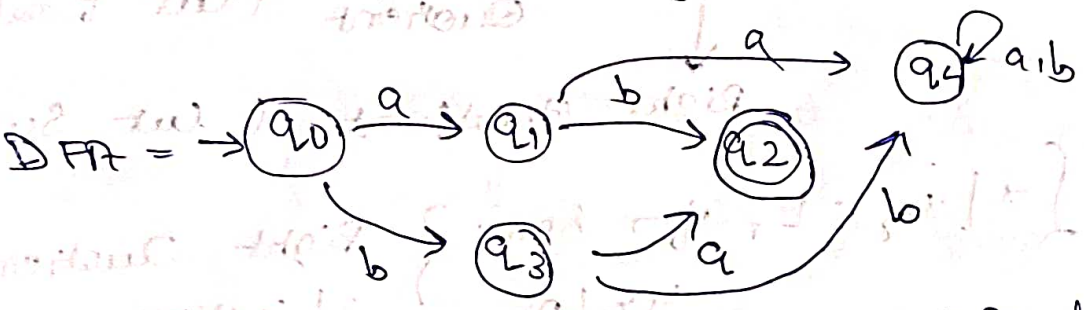
Let $L = \{ab, ba\}$

Prefix of ab = ε, a, ab

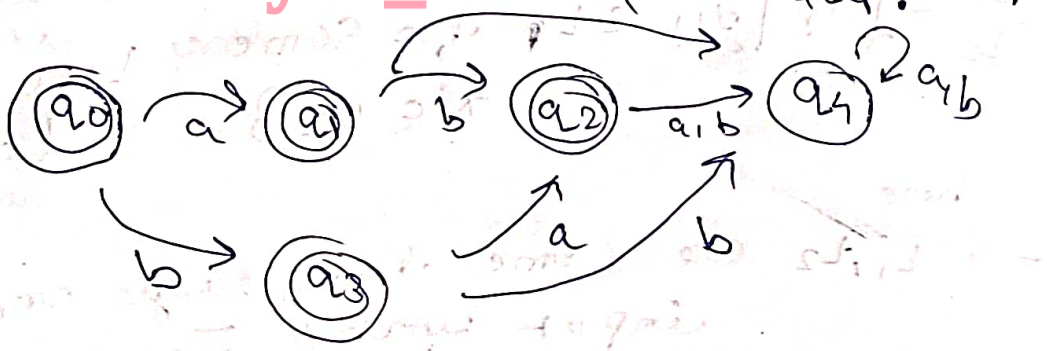
Prefix of ba = ε, b, ba

INIT of L

$= \{\varepsilon, a, ab, b, ba\}$.



For INIT DFA → Make all non-final states as final except trap.



Regular Languages are not closed under infinite Union.

DCFL - Deterministic context free lang.

CFL - context free Lang.

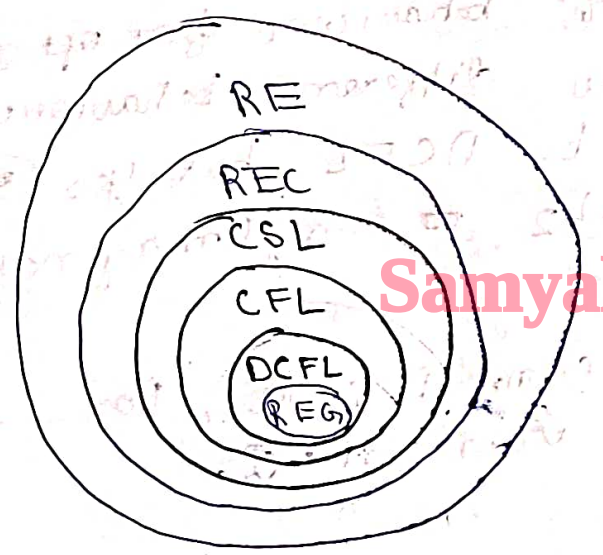
CSL - Context Sensitive Lang.

REC - Recursive.

RE - Recursive Enumerable.

	Regular	DCFL	CFL	CSL	REC	RE
$\cup$	Yes	No	Yes	Yes	Yes	Yes
$\cap$	Yes	No	No	Yes	Yes	Yes
L^c	Yes	Yes	No	Yes	Yes	No
•	Yes	No	Yes	Yes	Yes	Yes
*	Yes	No	Yes	Yes	Yes	Yes
+	Yes	No	Yes	Yes	Yes	Yes

* A Recursive enumerable language is only accepted by a turing machine.
They are countably infinite languages.



RE = Recursive enumerable
REC = Recursive languages. (Accepted by the halted turing machine)

* Linear bound automata accepts CSL.

DCFL ⊂ CFL (Context Free language).
↳ It is accepted by PDA

REG = Regular Languages (accepted by finite automa)

FA + 1 = PDA $\xrightarrow{+1}$ TM (Turing Machine)

REG → Type 3
CFG → Type 2
CSG → Type 1
Unrestricted Grammar → Type 0

NTM ≡ DTM
DPDA < NPDA
DFA ≡ NFA

FA < DPDA < RDA < LBA < TM

REG ⊆ DCFL ⊆ CFL ⊆ CSL ⊆ REC ⊆ RE

Which of the following is/are false?

(I) $\{a^n b^n\} \cup \{a^n b^{2n}\}$ is CFL but not DCFL.

(II) $a^m b^m c^n d^n$ is CFL but not DCFL.

(III) $\{a^m b^n \mid m, n \geq 0\}$ is DCFL but not regular.

Solution

(I) We have two diff. behaviour for opt. I $a^n b^n$ and $a^n b^{2n}$ show different behaviours, so they cannot be DCFL but yes CFL. Also $DFCL_1 \cup DFCL_2$ may/may not be DFCL.

(II) For every a, we can pop (a), For every c, we can pop (c).

Comparsions = 1

(b to a, b ; c, d)

So, it is an DCFL.

III It is regular.

Ex) $a^n b^n c^n$

1 1 = comparsions.

Total comparisions = 2

So, it is neither DCFL nor CFL but bcoz 2 stacks are required for 2 comparisions respectively.

$a^m b^n c^n d^m$ is also DCFL

for $a^m b^n$, $m = n$

then, it is not regular.

but DCFL.

(a) L(R) = Language represented by Regular expression, L(G) = " by context free grammar, L(D) = " by deterministic context free grammar. which is false?

(a) Given a regular expression R and a string w, is $w \in L(R)$ → true.

(b) Given a context free grammar G is L(G) = finite or infinite, is decidable - True

(c) Union of two L(D) language is closed. - False

(d) complement of L(G) language is not closed. - true

DCFL ∪ DCFL = CFL = not closed

LC = CSL = not closed

so,

(1) consider $r = (11 + 111)^*$ over $\Sigma(0,1)$. Find no. of states in minimal, NFA and DFA respectively.

→ N-3, D-4.

→ context free grammar is not closed under complementation.

2) The context free grammar

$S \rightarrow SS \mid 0S1 \mid 1S0 \mid \varepsilon$ generates:

$S \rightarrow SS$

$0S1 \quad 1S0$

$\varepsilon \quad \varepsilon$

0110 → equal no. of 0s and 1.

Q $A_0 \rightarrow aA_1$

$A_1 \rightarrow ab$

$A_0 \rightarrow aAb$

= regular grammar.

= Type 3 ;

CFG is given,

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

Generate, CFL

Homomorphism → Substitution fn.

$h(L) = \{h(\omega) \mid \omega \in L\}$

where $h: \Sigma$ to Γ^* is called homomorphism

$\Sigma\{0,1\}$ $\Gamma = \{a,b\}$

$h(0) = aa$, $h(1) = bb$

if $L = \{00, 101\}$

$h(\omega)$ = homomorphic image

Find $h(L) = \{$ **aaaa . bbaabb** $\}$ →

Just Substitute aa where you find 0 and bb, for 1.

here RE = $(0+1)^* 1^*$

$= (aa + bb)^* bb^*$

A homomorphic expression made from RE is also RE

So, RE can be converted into homomorphism this way.

Inverse Homomorphism $h^{-1}(L)$

Let $h(0) = a$, $h(1) = b$, $h(2) = ab$

$\Sigma = \{0,1,2\}$, $\tau = \{a,b\}$

Let $L = \{abab\}$

then $h^{-1}(L)$

= 01'01', 22, 201 and so on.

$h(h^{-1}(L)) \subseteq L$ → Remember.

Decidability Table

	REG	DCFL	CFL	CSL	REC	RE
Membership $w \in \Sigma^*$	✓	✓	✓	✓	✓	✗
Infiniteness Problem L = finite or infinite	✓	✓	✓	✗	✗	✗
Emptiness Problem. $L = \phi$	✓	✓	✓	✗	✗	✗
Equality Problem $L_1 = L_2$	✓	✓	✗	✗	✗	✗
L is Ambiguous	✓	✓	✗	✗	✗	✗
Completeness. $L = \Sigma^*$	✓	✓	✗	✗	✗	✗
$L_1 \cap L_2 = \phi$	✓	✗	✗	✗	✗	✗
If $L_1 \subseteq L_2$ Subset Problem.	✓	✗	✗	✗	✗	✗

Here 'V' = Decidable and 'X' means Undecidable.

REG = Regular Language,

here Ambiguous means whether a particular string can be generated by multiple grammar or not.

CFG → Context Free Grammar. (Type 2)

(V, T, P, S) → Start Variable

- V → Finite set of Variables (Non-terminal)
- T → Finite set of Terminals ($V \cap T = \phi$)
- P → Production rules (Substitution rules)

Variables → CAPITAL LETTERS

Terminal → Small case letters.

ex) $\alpha \rightarrow \beta$

α ⊕

For terminating, use ϵ.

ex) $S \rightarrow 0S1 \mid \epsilon$

here $\{0, 1, \epsilon\}$ are terminals.

So, $L = \{0^n 1^n, n \geq 0\}$

Convert CFL → CFG

1) $a^n b^n$, $n \geq 0$

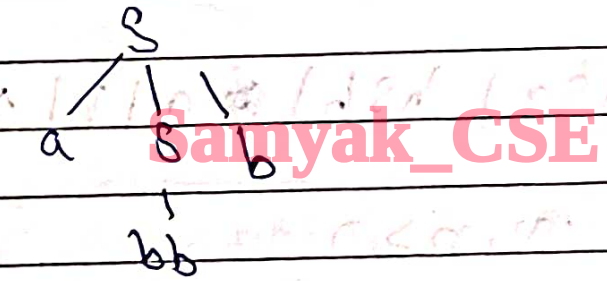
$S \rightarrow aSb \mid \lambda$

2) $a^n b^n$, $n \geq 1$

$S \rightarrow aSb \mid ab$

3) $a^n b^{n+2}$, $n \geq 0$

$S \rightarrow aSb \mid bb$



a bbb

4)

$a^{2n} b^n$, $n \geq 0$

$S \rightarrow aaSb \mid \lambda$

$a^{2n} b^n$, $n \geq 1$

$S \rightarrow aaSb \mid aab$

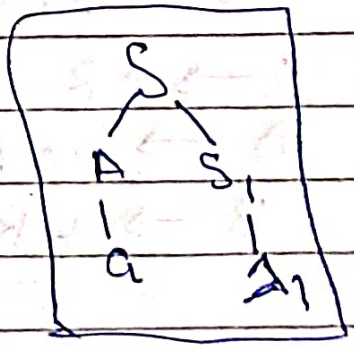
5) $a^m b^n$, $m > n$

$n \geq 0$

$S \rightarrow AS_1$

$S_1 \rightarrow aSb \mid \lambda$

$A \rightarrow aA \mid a$



RBDS

6) $a^{2n+3} b^n$, $n \geq 0$

$S \rightarrow aaSb \mid aaa$

For $a^m b^n$, $m \geq n$

$m, n \geq 0$

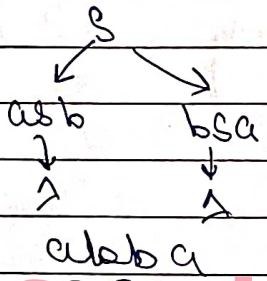
$S \rightarrow AS_1$

$S_1 \rightarrow aS_1b \mid \lambda$

$A \rightarrow aA \mid \lambda$

7) $\{ w \mid n_a(w) = n_b(w) \}$

$S \rightarrow aSb \mid bSa \mid SS' \mid \lambda$



8) $ww^R \cup w(a+b)w^R$

$S \rightarrow aSa \mid bSb \mid a \mid b \mid \lambda$

9) $a^m b^n c^n$, $m, n \geq 0$

$S_1 \rightarrow aS_1b \mid \lambda$

$C \rightarrow cC \mid \lambda$

$S \rightarrow S_1C$

Derivation

LMD

RMD

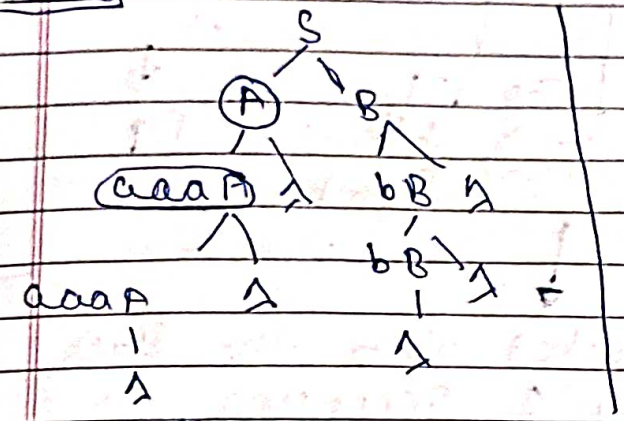
$S \rightarrow AB$

$A \rightarrow aaA \mid \lambda$

$B \rightarrow bB \mid \lambda$

Check whether aaaaabb ∈ L(G).

LMD

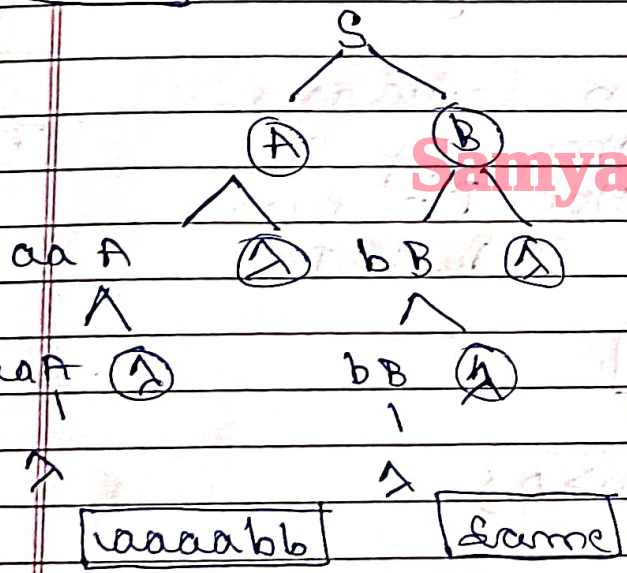


Derivation Tree for LMD (left Most Derivation)

= aaaAbb

Yes this ∈ L(G).

RMD



Derivation Tree for RMD (Right most Derivation).

aaaaabb | Same

Pushdown Automata

$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

Q = Finite set of states like F.A

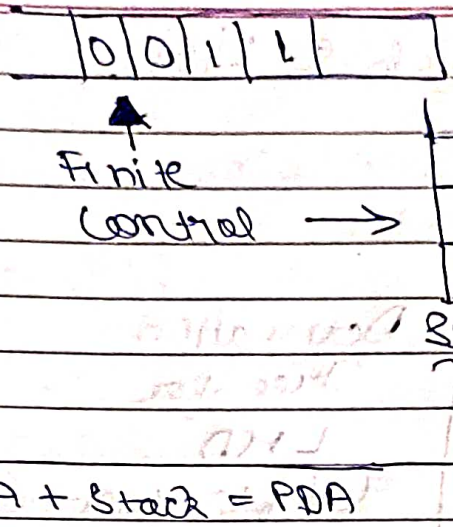
Σ = input symbol like F.A

Γ = Stack Alphabet (Push into Stack)

δ = Transition fn $\delta : Q \times (\Sigma \cup \varepsilon) \times \Gamma \rightarrow Q \times \Gamma^*$

q_0 = Initial State

= Final state, Z_0 = Stack start Symbol.



FA + Stack = PDA

'0' is pushed into the stack, now, for every '1' occuring we pop '0' from the stack, hence comparisions are easily performed.

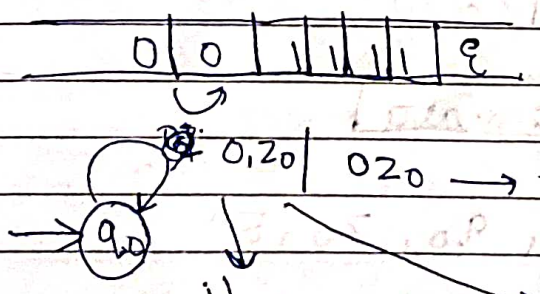
* PDA is used to recognise CFL.

{ When $\Gamma = \Gamma^*$ then unchanged.
When $\Gamma^* = \epsilon$ then pop Γ.
When $\Gamma \neq \Gamma^*$ then Γ = push.
(Push Γ).

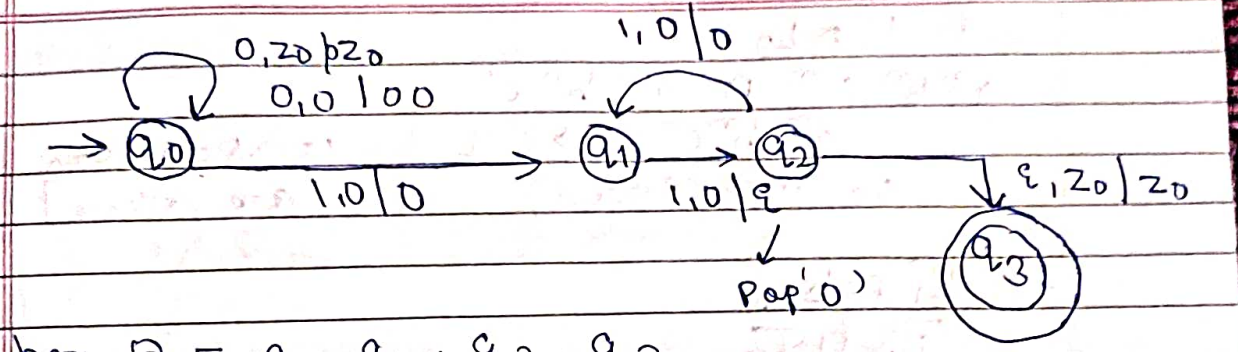
PDA Designing

ex)

$L = \{0^n 1^{2n}, n > 0\}$



→ q_0 0 Z_0 → then keep Z_0, then push 0, so we have $0Z_0$.



here $Q = q_0, q_1, q_2, q_3$

$\Sigma = 0, 1$

$\Gamma = 0, Z_0$

δ : $q_0, 0, Z_0 \rightarrow 0Z_0$

$q_0, 0, 0 \rightarrow 00$

$q_0, 1, 0 \rightarrow q_1, 0$

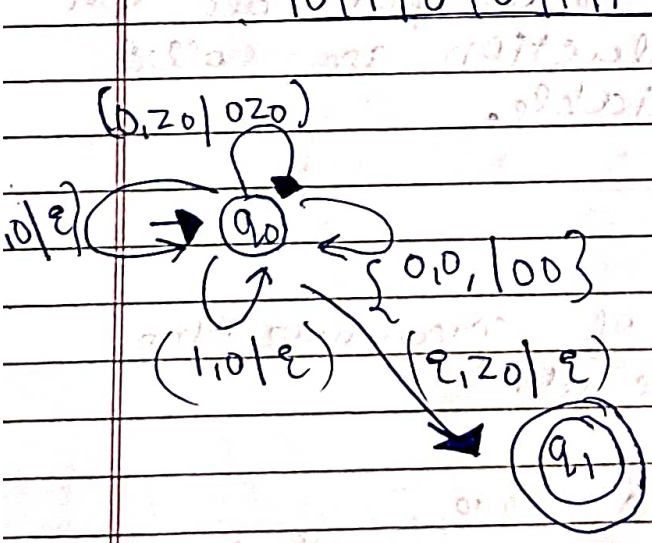
$q_1, 1, 0 \rightarrow q_2, \varepsilon$

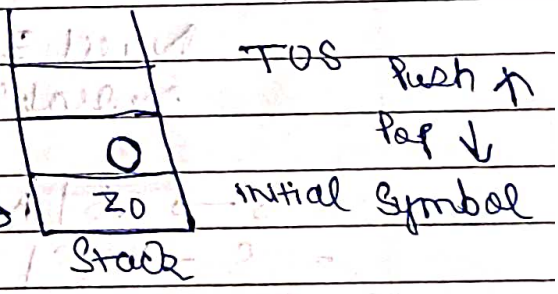
$q_2, 1, 0 \rightarrow q_1, 0$

$q_2, \varepsilon, Z \rightarrow q_3, Z_0$

PDA design for $L = \{ w \in (0,1)^* \mid n_0(w) = n_1(w) \}$

0	1	0	0	1	1





Closure Properties of CFL

Union ✓ Concatenation ✓

Kleene Closure ✓ Intersection ✗

Complementation ✗

Ex) $L_1 \cap L_2$

$a^n b^n c^m \cap a^m b^n c^n$

$a^n b^n c^n \rightarrow$ CSL [2 comparisons are there]

So not closed.

→ $L_1 \cap L_2$

= $L_1 \cup L_2$

= FD ∩ FL

= not closed

Elimination of Null (ε) Production

Ex-1) $S \rightarrow aS \mid A$

$A \rightarrow \varepsilon$

↓

nullable Variables

{A, S}

The variables which are responsible for Null(ε) production are called nullable variable.

$S \rightarrow aS \mid A \mid a \mid \varepsilon$

$S \rightarrow aS \mid a \mid \varepsilon$

If 'ε' is a part of grammar then, we cannot remove it.

$a^* a^n$, $n \geq 0$

If ε is removed here, then L(G) changes to $a^* a^n$, $n \geq 1$

Ex-2) $S \rightarrow ABC$

$A \rightarrow aA \mid \varepsilon$

$B \rightarrow bB \mid \varepsilon$

$C \rightarrow c$

{A, B} → nullable Variable

$S \to ABC \mid BC \mid AC \mid C$

$A \to aA \mid \varepsilon \mid a$

[Put $A = \varepsilon$, $B \neq \varepsilon$, $AB = \varepsilon$]

$= A \to aA \mid a$

$B \to bB \mid a$

$C \to c$

"How to remove unit Production from CFG"

ex) $S \to AB$

$A \to a$

$B \to C \mid b$

$C \to D$

$D \to E$

$E \to a$

Unit Production are of type

$\alpha \to \beta$

$\alpha, \beta \in V$

↳ belongs to

here, Unit Production variables are $\{ B \to C, C \to D, D \to E \}$.

$S \to AB$

$A \to a$

$B \to a \mid b$

$C \to a$

$D \to a$

$E \to a$

we notic from Start Symbol 'S' that we are unable to reach C, D, E from Start So we can simply remove them.

=

$S \to AB$

$A \to a$

$B \to a \mid b$

Ex-2) $S \to aA \mid B$

$A \to ba \mid bb$

$B \to A \mid bba$

$S \to B$

$B \to A$

$S \to B \to A$

$= S \to aA \mid ba \mid bb \mid bba$

$A \to ba \mid bb$

reachable

B was not ∧ so removed

Turing Machine

Blank Symbol

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

$\delta = Q \times \Gamma \rightarrow Q \times \Gamma \times (L, R)$

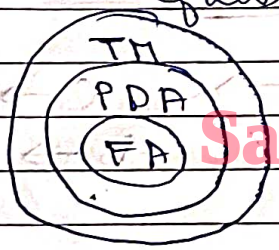
| B | a | b | B | | | |

$\Sigma \subseteq \Gamma$

R/W Head → I/P Tape

Finite Control → Read-Write Head (Both operation are allowed)

* When comparisons are more than 2, PDA fails, e.g. $a^n b^n c^n, n \geq 1$

→ 

In PDA, not from both the directions read is allowed but in TM, from both ends it is allowed.

Linear Bound Automata (LBA)

FA < PDA < LBA < TM

LBA = TM + Limited Size tape.

| $ | a | b | a | B | $ |

Standard Examples

1) $L = \{a^n b^n c^n ; n \geq 1\}$

2) $L = \{a^n : n \text{ is prime}\}$

3) $L = \{a^n : n \text{ is non-prime}\}$

4) $L = \{ a^{n!} , n \geq 0 \}$

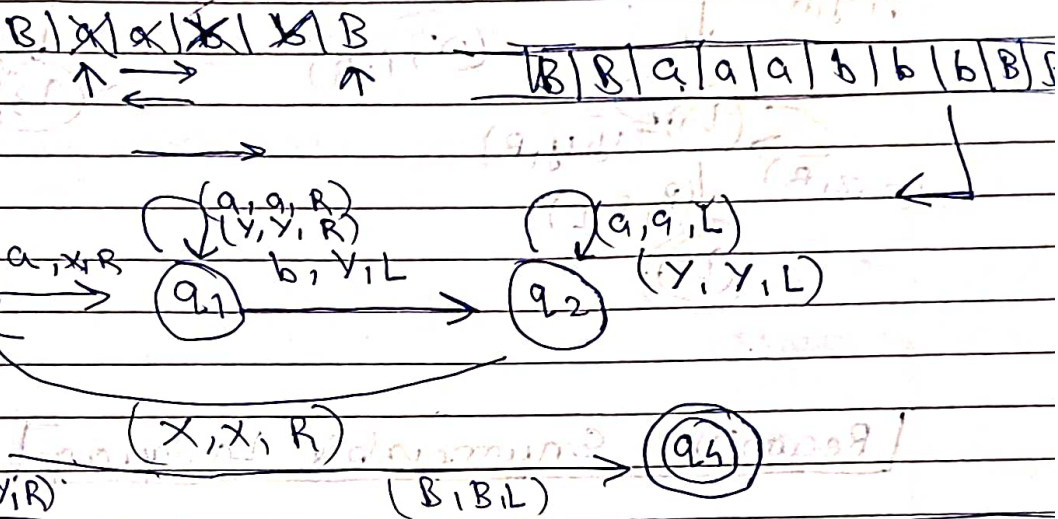
5) $L = \{ ww : w \in (a,b)^* \}$

6) $L = \{ www^R , : w \in (a,b)^* \}$

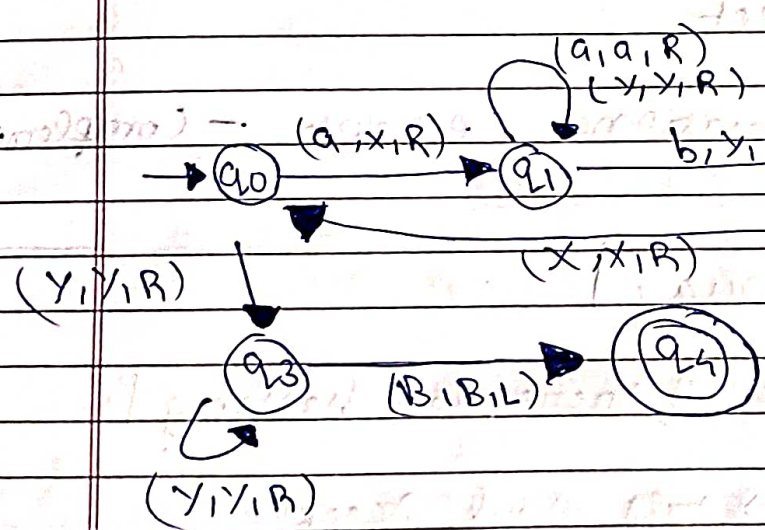
7) $L = \{ w^n : w \{a,b\}^* , n \geq 1 \}$

8) $L = \{ a^n : n = m^2 , m \geq 1 \}$

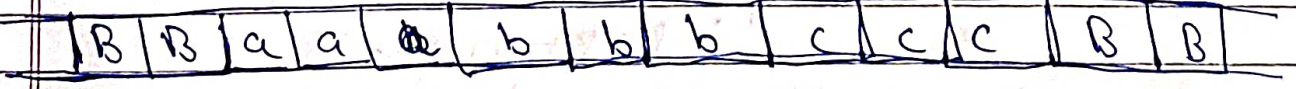
Turing Machine for $\{ a^n b^n \mid n \geq 1 \}$



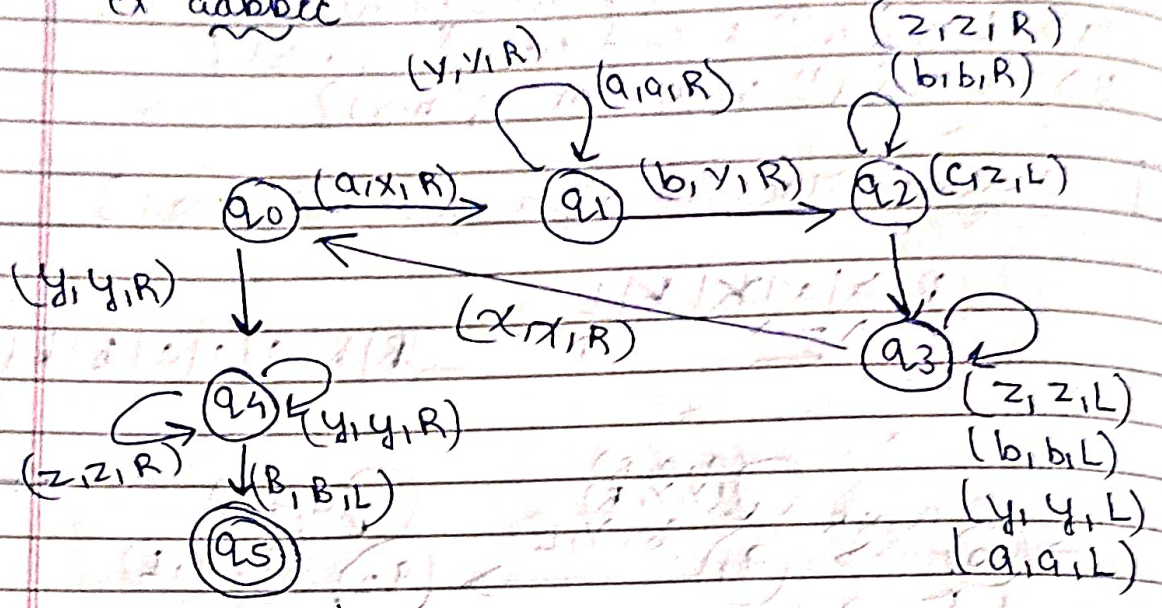
(a, X, R) means, when we observe 'a', convert it into X and move Right R.



② $\{ a^n b^n c^n \mid n \geq 1 \}$



Ex aabbcc



Recursive Enumerable Language — RE

* L is RE, if there is TM [Turing Machine]

→ Three States
Halt and Accept
Halt and Reject
Never Halt

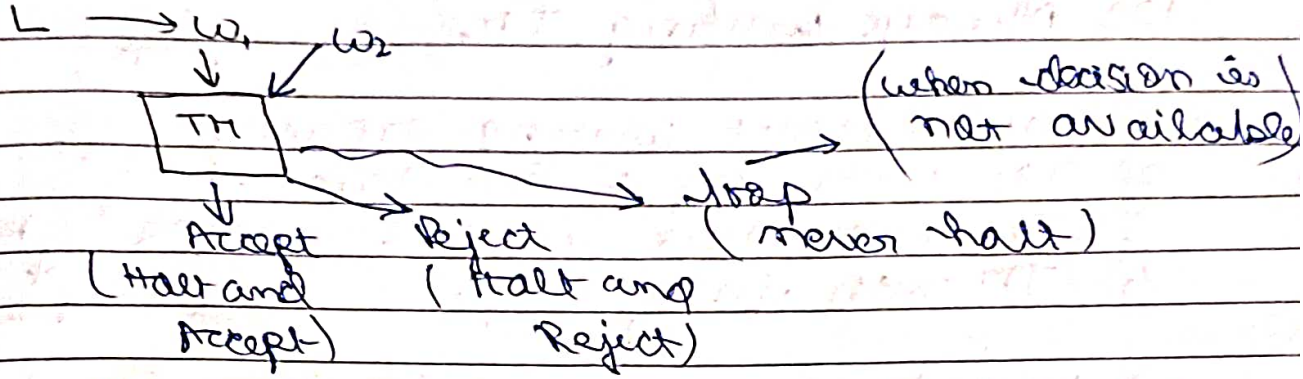
Closed under all operations except — Complement

Recursive language

→ L is recursive if there is halting / Total TM

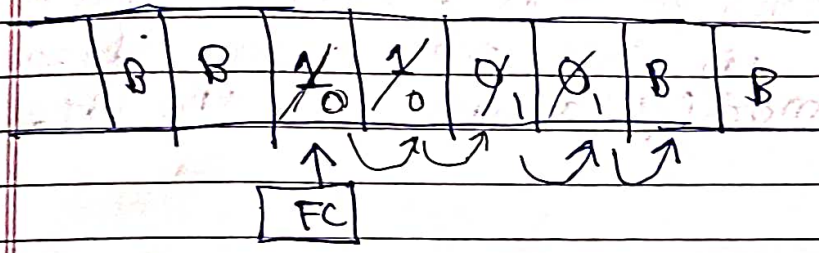
Two States → Halt and Accept
→ Halt and Reject.

→ Closed under all except Homomorphism and Substitution.



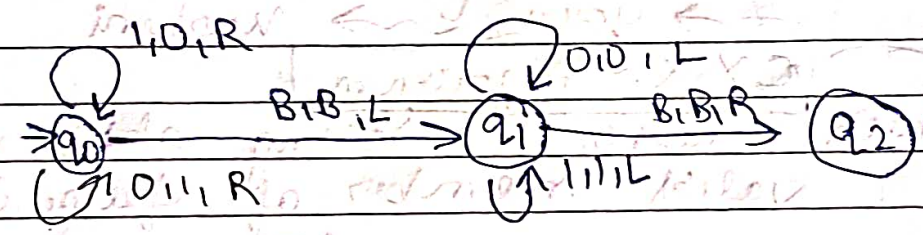
Turing Machine for 1's complement

It converts input to output



States	0	1	B
q_0	$1Rq_0$	$0Rq_0$	BLq_1
q_1	$0Lq_1$	$1Lq_1$	BRq_2
q_2	—	—	—

new state, R, L, Change Direction



Turing Machine with modifications

1) TM with Stay option
2) TM with Semi-infinite tape
3) Offline TM
4) Multidimensional TM
5) NDTM → non-determ. TM
6) UTM → Universal TM
7) Multitape TM
8) Multihead TM

9) TM with 3 stacks.
10) Jumping TM
11) Non erasing TM
12) Always writing TM.

1) TM without writing capacity → (FA)
2) TM with input size tape → (LBA)
3) TM with tape used as stack * (NDT)
4) TM with finite tape (FA) — PDA (DDTM)

→ More languages accepted = More Powerful T

* 1 → 12 types of TM has no-effect on power of TM however Time complexity is not same for these (1-12) modifications in the TM.

(1 → 4) →

(2) → is most powerful than others.

FA < PDA < LBA < TM

DPDA < NPDA

younger → Kashmiri

CYK Algorithm

Check whether a string abbb is a valid member of following CFG.

→ S → AB
A → BB | a
B → AB | b

→ CYK applicable on CNF only.

CNF: A → BC
A → a OR

* It is universal (applicable on all CNF)

* Time complexity $O(n^3)$
* Space complexity $O(n^2)$

* CYK is a algorithm belonging to membership algorithm.

ex1 abbb
1234

Given CFG, convert it into CNF, then apply CYK algorithm.

	4	3	2	1
1	S,B	A	S,B	A
2	S,B	A	B	
3	A	B		
4	B			

First fill all the diagonal elements
See abbb
↓
{ a is derived by A, so write A.
abbb

1 → 3 Pair
abb
= 1 2 3
(11)(23) { Both are used)
R (12)(33)
↓
we take (11)(23)
↓ ↓
A A
↓
Φ | according to grammar given

Now, we check,
(12) (33)
↓ ↓
SB B
SB BB
↓ ↓
Φ A

we have 2 cases [AA and ΦA
AA ∨ ΦA = A
So, write A]

→ After the matrix is complete,
At first box, Starting symbol (S) is found thus, we conclude that string 'abbb' is a membership of the CFG.

CNF	GNF
(Chomsky Normal Form)	(Greibach Normal Form)
1) $A \to BC$ ($A, B, C \in$ Variables) OR $A \to a$ ($a \in$ Terminal)	1) $A \to ax$ where $\{x \in V^*, a \in T\}$
2) No. of steps required to generate a string of length n is $2\lvert n\rvert - 1$.	2) No. of steps required to generate a string of length (n) is n.
3) Used in Membership Algo.	3) Used to convert CFG to PDA
4) Derivation or Parse tree obtained from CNF is always binary Tree.	4) Derivation or Parse tree obtained from CNF is not always binary tree.
5) Length of each production is restricted.	6) Not Restricted.

Derivation Tree / Parse Tree in TOC

It is basically the sequence of steps followed to generate a word (string) from a given grammar.

G: $E \to E+E \mid E*E \mid E=E \mid id$

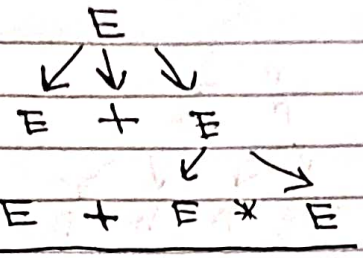
w: id * id + id * id

$w(S) \in L(G)$:

$E \to E+E$

$\downarrow$

$E+E*E = id+id*id$

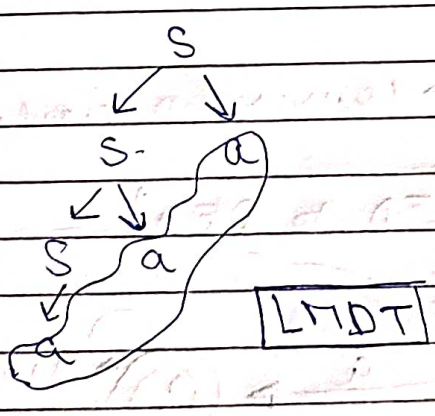


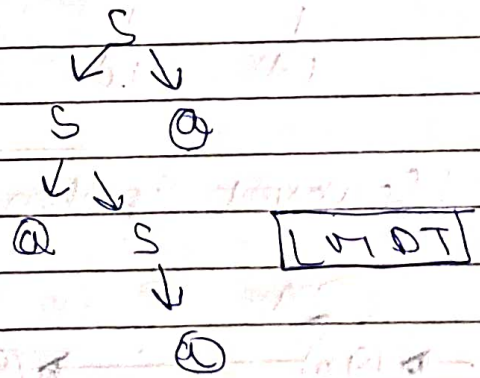
Recursive CFG generates infinite no. of strings	Non-recursive CFG generates finite no. of strings.
ex) $S \to Sa$, $S \to b$	a) $S \to Aa$, $A \to b \mid c$ ea

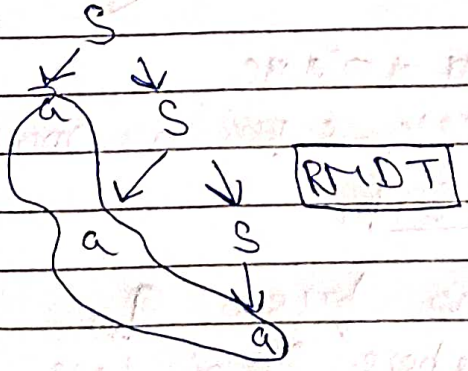
* For ambiguous grammar there exists more than one derivation for any word that belongs to grammar.
* For unambiguous grammar there exist exactly one derivation for any word that belong to grammar.

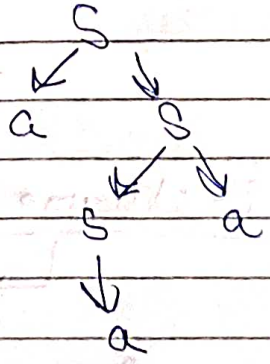
For example: Given that G: $S \to Sa \mid as \mid a$

ex: ω: aaa

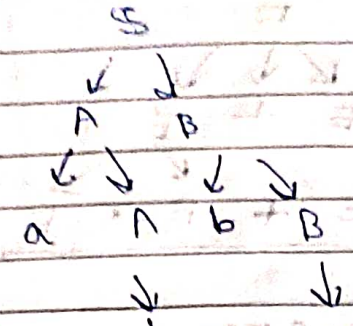








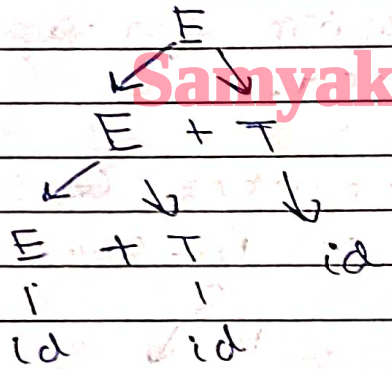
Gr1: S → AB
A → aA | b
B → bB | a
ω: abba



① Convert the Ambiguous grammar into unambiguous one.

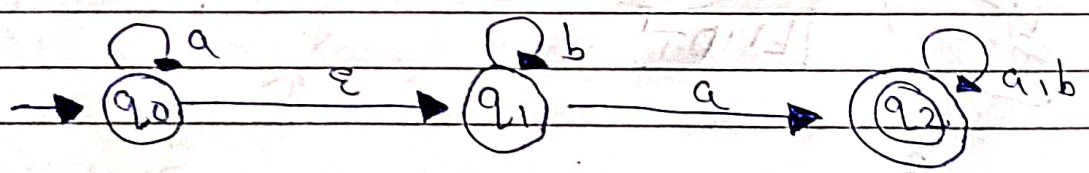
→ G1: E → E+E | id
ω = id+id+id

↳ E → E + T | T
T → id



→ conversion done.

Convert epsilon ε-NFA to DFA



q0 → initial state.

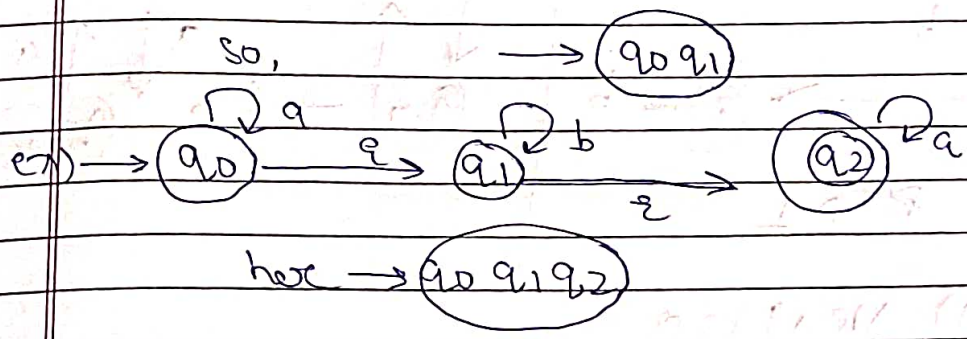
eliminate e-psilon moves

ε-closure of $q_0 = \{q_0, q_1\}$

↳ q_1 means from q_0 using ε- where can we reach.

ε closure of $q_0 = q_0 q_1$

So, $\rightarrow (q_0 q_1)$



here $\rightarrow (q_0 q_1 q_2)$

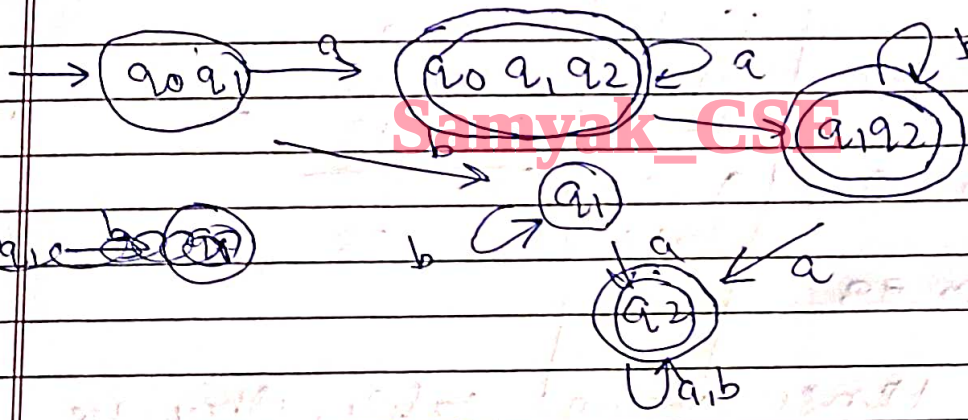
$\varepsilon a = a$

using both ε, a we can reach q_1 from q_0

ex-1 $\rightarrow$ $\rightarrow (q_0 q_1)$ $\xrightarrow{a}$ $q(q_0) \cup a(q_1)$

$= q_0 q_1 q_2 \cup q_2$

$= q_0 q_1 q_2$



q_2 was final in NFA so in DFA also q_2 states was made final

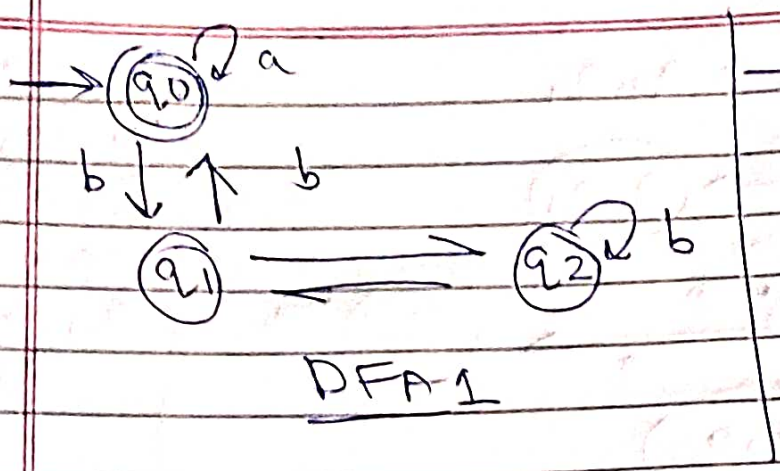
(Ans)

{ In NFA, here states = 3, so max^m possible states present in DFA = $2^3 = 8$
here, DFA has 5 states. }

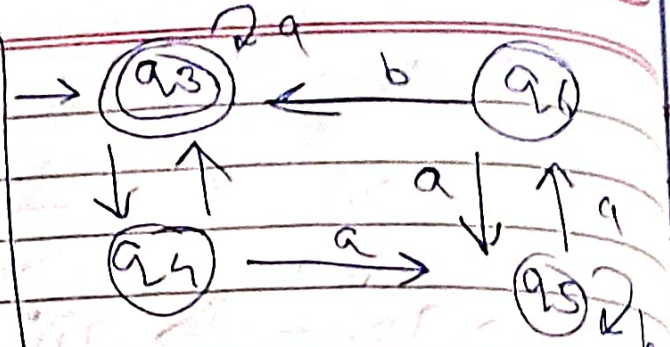
To check if two DFA's are equivalent.

(Step-1) Check if their initial state and final state are same or not.

(Step-2) Check compare the transitions of their initial states with the Σ (input variables.



DFA 1



Step 1) Verified

Step 2)

	a	b
	q_0 q_3	q_1 q_4
	FS FS	IS IS
	Same	Same.
q_1 q_4	q_2 q_5	q_0 q_3
	IS IS	FS FS
q_2 q_5	q_1 q_6	q_2 q_5
	IS IS	IS IS
q_1 q_6	q_2 q_5	q_0 q_3
	IS IS	FS FS
~~q_0 q_3~~		

IS = Intermediate State

Hence from table, they are equivalent, so, Step-2 is verified.

Hence DFA 1 and DFA 2 are both equivalent.