THEORY OF COMPUTATION

→ Language Processing.

→ Compiler → Converts HLL to machine language.
→ language accepter / translater.
language → Set of strings processed by compiler.
Grammar → Set of rules to generate strings for language.

→ Production rule of grammer.

G_1 {
$S \to AB$
$A \to aA / \lambda$ → NULL.
$B \to bB / \lambda$

→ String is a set of terminals (small letter) ~~String consists~~
non-terminal / variable (capital letter).
Terminals (small letter) Non-terminal / variable (capital letter)
→ String consists of only capital letters

$S \to A\ B$
$aA\ B$
$a\ aA\ B$
$a\ a\ \lambda\ B$
$a\ a\ B \to a\ a\ b\ B \to aab\lambda$
$a\ a\ b$

} Sentential Form.
→ Generating string from grammar.

→ $L_1 = \{\lambda, a, b, ab, aab, abb, \ldots\}$
$L_1 = \{a^n b^m \mid m, n \geq 0\}$

→ Non-terminal are used to generate strings while terminal are part of string.

→ $G_2 \{ S \to aSb / \lambda$ — aSb, λ

$aSb \to a\lambda b \to ab$

$aaSbb \to aabb$

$L_2 = \{ \lambda, ab, aabb, \ldots \} = a^n b^n \mid n \geq 0$

→ ↳ Number of a = Number of b and a is followed by b.

→ Cardinality of $\lambda = 0$

" of $a = 1$.

→ $G_3 \Rightarrow \{ S \to aSb / bSb / \lambda$

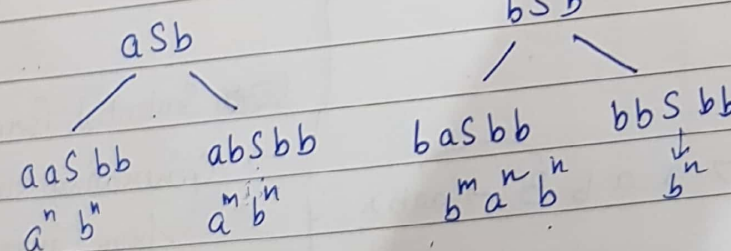
$L_3 = \{ \lambda, ab, bb, aabb, abbb, babb, bbbb, \ldots \}$

aSb ↓

$bSb \to bb$

$\to baSbb \to babb$.

$\to bbSbb \Rightarrow bbbb$



→ $G_4 = \{ S \to aSb / bSa / \lambda$

⇓

No. of a = No. of b

↳ Even length palindrome are generated by this grammer.

Book → Peter Linz

→ Chomsky Classification of grammar

– Classifies into 4 categories according to production rule.

① Type 0 (unrestricted G) $\alpha \to \beta$

α, β are set of terminals and non terminals strings.

② Type 1 (Context Sensitive G) $|\alpha| \leq |\beta|$ (Follows type 0)

③ Type 2 (Context free G) → $\alpha \to \beta$ (Follows type 0 + type 1)

$|\alpha| = 1$ and α should be non-terminal only

④ Type 3 (Regular G) → Follows type 0 + 1 + 2.

β can be $aA / Aa / a / \lambda$

Can be anything

It cannot be $A \to BB$.

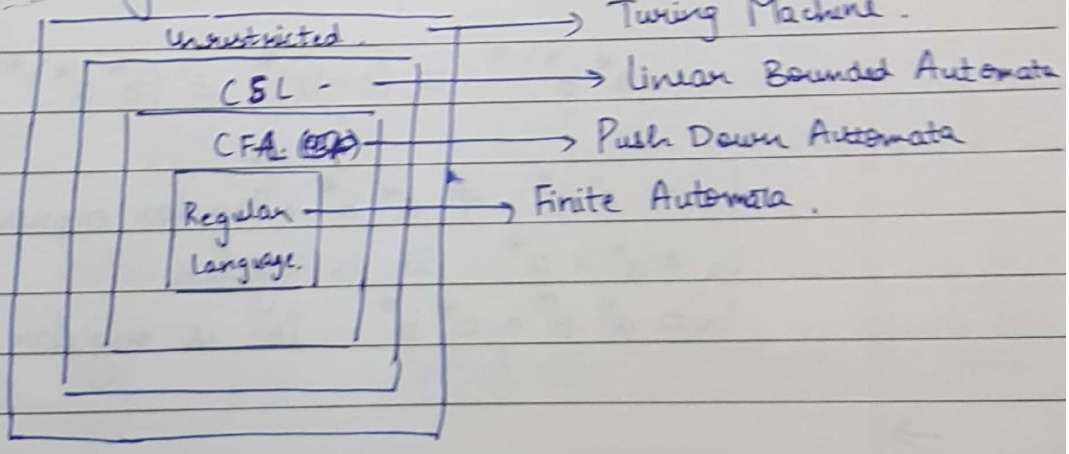
Type 3 ⊆ Type 2 ⊆ Type 1 ⊆ Type 0 (⊆ : subset)

→ For every category of grammar, there will be a corresponding language and machine.

→ Type 3 → DFA

Type 0 → Turing machine.

→



→ DFA / Regular Expression.

FA

Deterministic — Non-deterministic

→ DFA / Regular Expression → Regular expression used to represent regular language

$a^* = \{\lambda, a, aa, \ldots a^n\}$

$* = \{0 - \infty\}$.

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

$+ = \{1 - \infty\}$

$a^* b^* = \{\lambda, a, b, ab, aab, abb \ldots\}$

$a^* b^* \neq b^* a^*$

$a^+ b^+ = \{ab, aab, abb, \ldots\}$

$L = (a+b)(a+b)$, OR

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

$L = (a+b)^3$

$\sum_{2}^{n} = \{a, b\}$

→ Regular expression for all strings possible using a and b

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

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

$L_1 = a^n b^n \neq a^* b^*$, Regular expression (Not It cannot be done by DFA because it has no memory so it can't store count of a and b)

$L_2 = a^n = a^*$ It is regular.

$L_3 = a^n b^m = a^* b^*$ It is regular.

→

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

$q_0 \rightarrow$ Q/Assignate

Date
Page

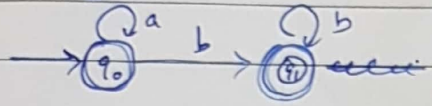
→ Finite Automata

– If it is possible to draw FA for any language, then language is regular.

– FA – Set of states in which there is one initial state and more than one final state.

DFA – Deterministic Finite Automata.

–



$\rightarrow L = \{ ab, b, a^n b, bb, \ldots \}$ $n \geq 0$

Initial State — Final State

$L = \{ a^n b \times (abb \text{ is also accepted}) \}$

NFA

$L = a^n b^m = a^* b^+$

$n \geq 0 \quad m > 0$

– In DFA, there should be transition for every input.

→ q_0 $L = \{\phi\} = \{\}$.

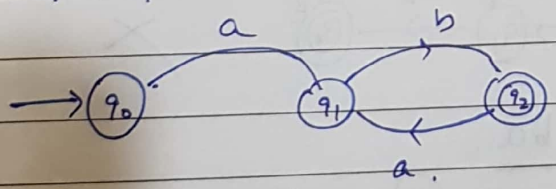
→ q_0 $L = \{\lambda\}$

$a.\lambda = a$

$ab.\lambda = ab$.

$a.\phi = \phi$.

→

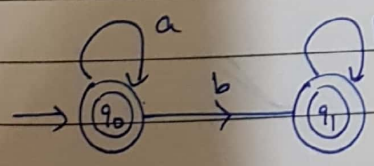


NFA

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

$= (ab)^+$

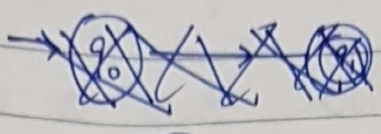
→

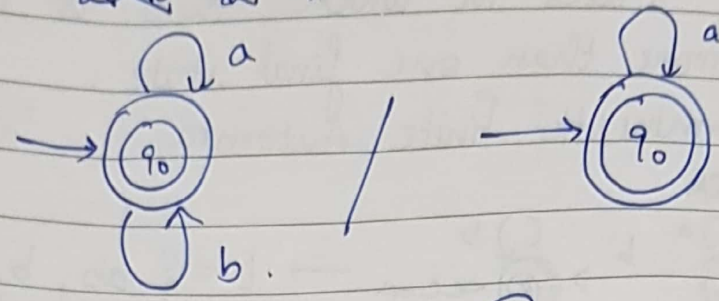


Initial state represented by arrow.

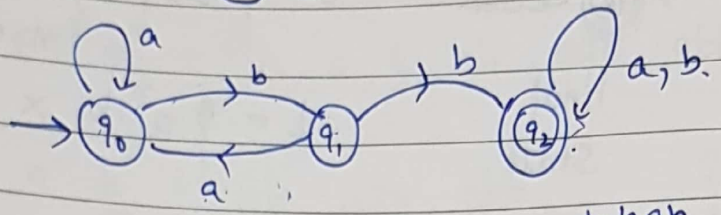
$L = \{\lambda, a, b, \ldots\} = a^* b^*$.

$L = \{a+b\}^*$



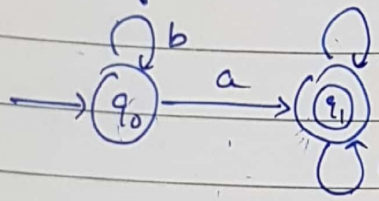


→



$L = \{ abba, abbb, bbab, bbbb, babb, bbaa, bbba, aabb \}$

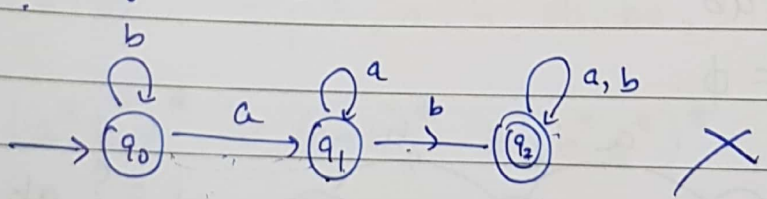
→ ① Design DFA - for the language containing atleast one a.



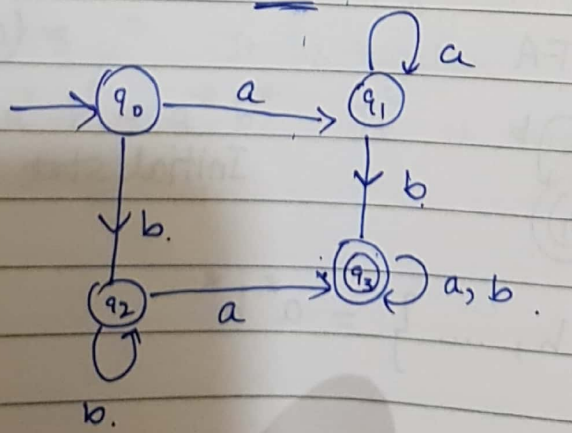
Every state should have one transition for exactly one input

① Design DFA for language containing atleast one a and one b.

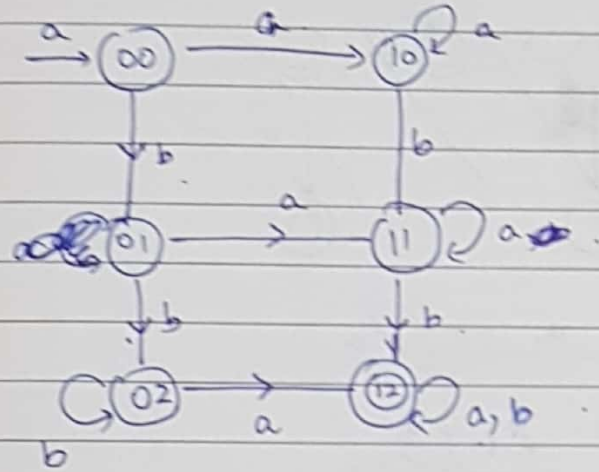
→



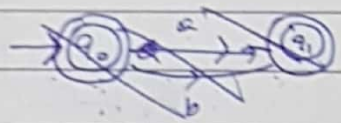
→ ba.



Q. Design FA with atleast one a and two b.



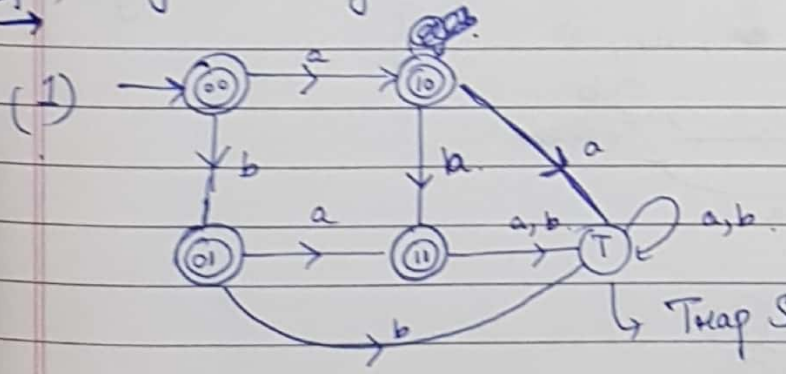
Q 1) Atmost one a and b



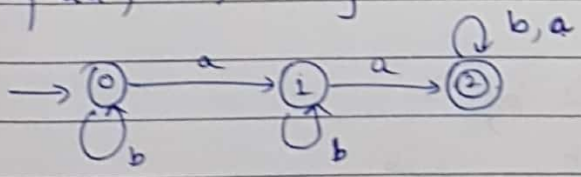
$L = \{ \lambda, a, b, ab, ba \}$

Q 2) Design DFA for atleast two a.

→ (1)

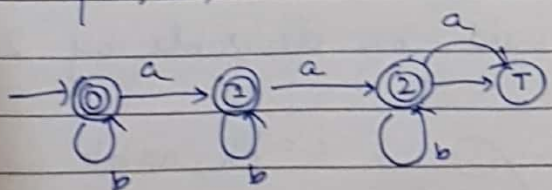


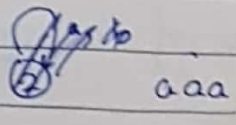
(2) $L = \{ aa, aba, \ldots \}$



→ (3) Atmost 2 a.

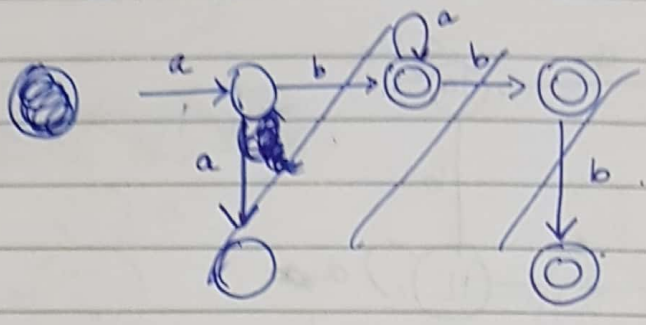
$L = \{ \lambda, a, $

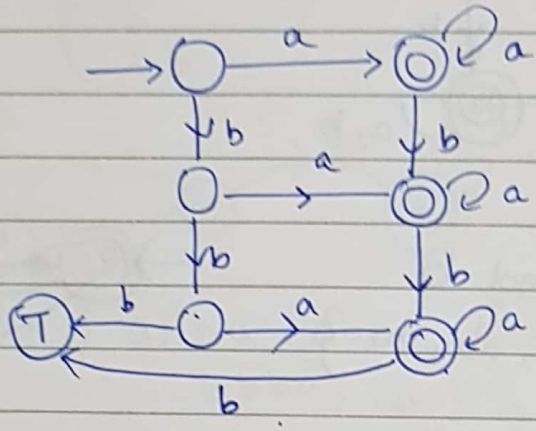




④ Design DFA with atleast one a and atmost 2 b.

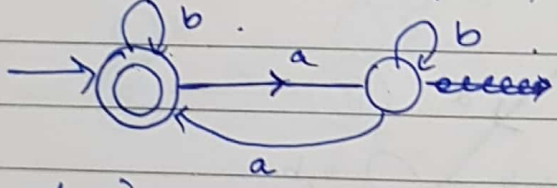
→



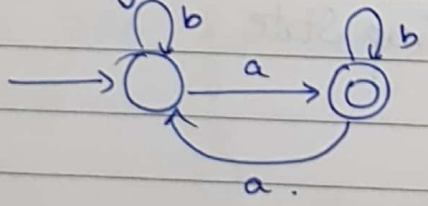


⑤ Design DFA in which number of a divisible by 2.

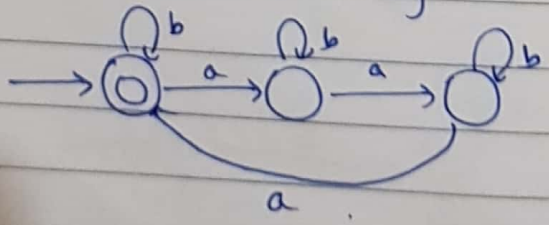
$L = \{ \lambda, b, aa, bb, \ldots \}$



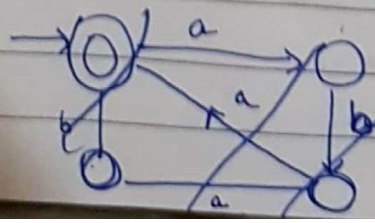
⑥ Odd no. of a's



⑦ No. of a is divisible by 3



⑧ No. of a and b should be divisible by 2.



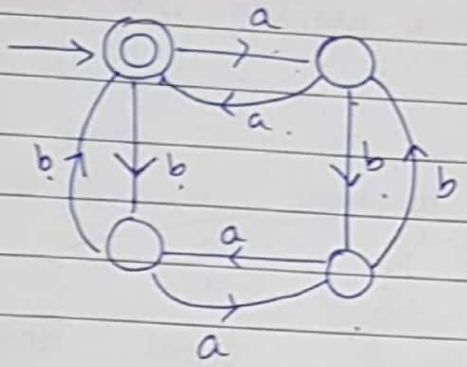
$L = \{ \lambda, aabb$

aab ✗

~~a* b*~~

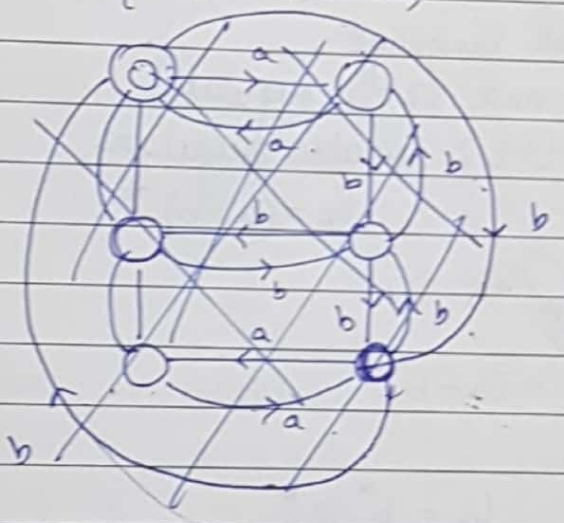
aabbb

$n(a) \% m = 0$ and $n(b) \% l = 0$. then no. of states $= m \times l$.

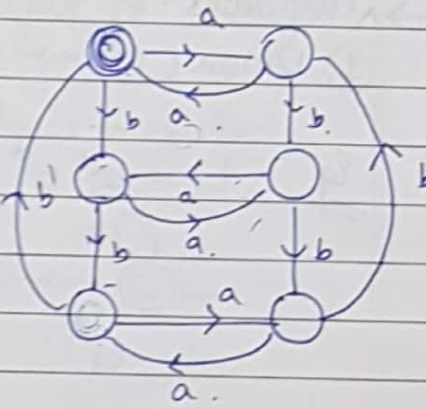


(9) No. of a should be divisible by 2 and no. of b is divisible by 3.

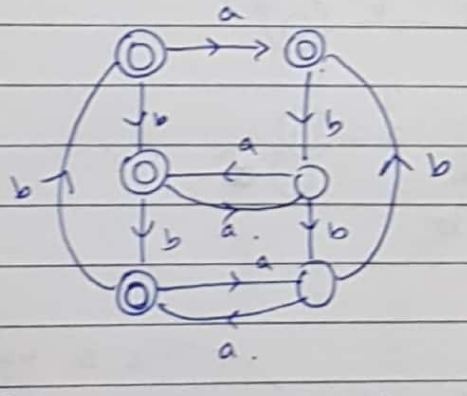
$L = \{\lambda, aa, bbb, \ldots\}$



(10)



→ (10) $n(a) \% 2 == 0$ or $n(b) \% 3 == 0$



→ $L = a^n b^n$ $n \geq 0$ No. of strings → Infinite

↳ Not regular

Theorem. $L = a^n b^n$, $n \leq 3$ No. of strings → Finite.

↳ For finite no. of strings, NDFA is possible to make. Then it is regular

$(A \cup B)^c = A \cap B$

→ Every finite language is regular. But it's reverse is not true.

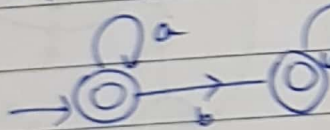
→ DFA / NFA → 5 tuple machine

- Q (Set of states)
- Σ (Set of input)
- δ Transition function
- q_0 Initial State.
- f Final state.

→ Properties of regular language. (closure)

① Union (∪) → L_1 and L_2 is regular
$L_1 \cup L_2$ is also regular

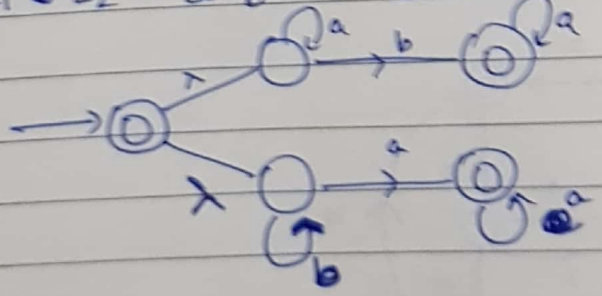
Eg $L_1 = a^* b^*$ $L_2 = (a+b)^*$



$L_1 \cup L_2 = (a+b)^*$

Eg $L_1 = a^* b^*$ $L_2 = b^* a^*$.

$L_1 \cup L_2 = a^* b^* + b^* a^*$.



→ Infinite Union

$\{a, b\} \cup \{aa\ bb\} \cup \{aaa\ bbb\} \ldots\ldots$ → This is regular.

$= a^n b^n$

↳ This is not regular.

$a^n b^n$ ✗ $a^* b^*$ 1 to ∞ $a^n b^n$ $\{a\} \cup \{ab\} \cup \{aabb\} \cup \ldots$ $\{a\} \cap \{b\} \cap \ldots$

→ $\{b\} \cup \{bb\} \cup \{bbb\} \cup \ldots = b^n$
It is regular.
Infinite union may or may not be regular. Therefore, regular language is not closed.

→ Regular Union Non Regular.
$a^* b^* \cup a^n b^n = a^* b^*$
R NR ↳ Regular.
$\phi \cup a^n b^n = a^n b^n$
R NR ↳ Non-Regular.

(2) Intersection ($\cap$)
→ L_1 and L_2 are regular ⇒ $L_1 \cap L_2$ is regular.
e.g. $L_1 = a^* b^*$ $L_2 = b^* a^*$
$L_1 \cap L_2 = a^* + b^*$.

→ Infinite Intersection
$\{ab\} \cap \{aabb\} \cap \{aaabbb\} \cap \ldots\ldots$
$= \phi$ (regular).

Assignment
Q(1) Whether regular language are closed under infinite intersection?

→ Regular Intersection Non-Regular.
$a^* b^* \cap a^n b^n = a^n b^n$
R NR NR.
$\phi \cap a^n b^n = \phi$
R NR R.

(3) Subset ($\subseteq$)
→ $L_1 = a^* b^*$
$L_2 = a^n b^n \subseteq L_1$
↳ It is not regular
Therefore, regular language are not closed under subset.

→ Every non-regular language has a regular subset (ϕ) and regular superset $(a+b)^*$.

→ Every non-regular language is infinite whereas every regular language can be finite or infinite.

④ Concatenation

→ $L_1 = \{ab, bab\}$
$L_2 = \{bb, ba\}$

$L_1 . L_2 = \{abbb, abba, babbb, babba\}$

$L_1 . L_2$ is always regular.

→ Regular language is closed under concatenation.

$L_1 . \lambda = L_1$

$L_1 . \phi = \phi$.

→ ⑤ Commutative.

$L_1 . L_2 \neq L_2 . L_1$

Concatenation is not commutative.

$L_1 \cup L_2 = L_2 \cup L_1$

$L_1 \cap L_2 = L_2 \cap L_1$

→ Regular. Non-Regular ⇒ Regular / Non-Regular

$\lambda . a^n b^n = a^n b^n$ (NR)

$\phi . a^n b^n = \phi$ (R)

⑤ Kleens closer (Power of a set)

→ $L^0 = \{\lambda\}$
$L^1 = \{ab\}$
$L^2 = \{abab\}$
$L^3 = \{ababab\}$
.
.

$L^* = L^0 \cup L^1 \cup L^2 \ldots \infty = \{(ab)^*\}$

Kleen's closure of a regular language is always regular.

→ $L = \{\lambda\} \Rightarrow L^* = \{\lambda\}$.
$L = \phi \Rightarrow L^* = \{\lambda\}$.

λ is always there in Kleen's closure.

→ Assignment

Q.2) Kleen's closure of non regular lang. can be regular or non-regular. Justify with example.

Q.3) $L = a^p$ where p is prime number.
Find the Kleen's closure. Also check whether it's DFA is possible.

→

⑥ Complement (L^c)

→ All strings that are not part of that language

$L = a^* b^*$

$L^c = (a+b)^* - L$.

→ $L = (a+b)^*$

$L^c = \phi$.

Regular language are closed under complement.

→ $L = a^n b^n$

$L^c = (a+b)^* - a^n b^n$ → Non Regular.

(Because checking for inequality also requires memory).

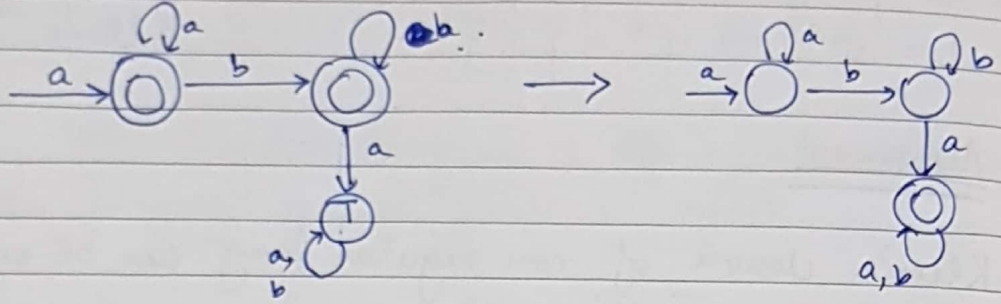
Complement of non-regular is always non-regular.

→ Steps for making DFA of complement of a language.

① - Design DFA of language whose complement has to be designed including trap state.

② Make final states to non-final and non-final states to final.

$L = a^* b^*$



⑦ Transpose

→ Transpose of regular language.

$A \rightarrow T^*B$.

or

$A \rightarrow T^*$.

Linear Grammar

① Right linear

$A \rightarrow aB$

Non-terminal after terminal.

$S \rightarrow aA \mid bB$

$A \rightarrow a$

$B \rightarrow b$

② Left linear.

$A \rightarrow BT^* / A \rightarrow T^*$

Non-terminal on left.

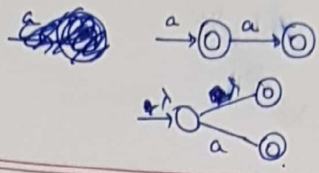
→ Language generated by left/right linear grammar is definitely regular but language generated by linear grammar may or may not be regular.

→ For finding transpose of L:

① Write grammar of L

② Reverse it's variable/non terminal.

→ Regular language are closed under transpose.



→ Non-Standard Operations.

① If L_1 and L_2 are regular

$L_1 - L_2$ returns a lang which is part of L_1 but not part of L_2.

$L_1 - L_2 = L_1 \cap L_2'$

→ Regular language is closed under this property.

② Reverse

→ $L_1 = \{ab\}$

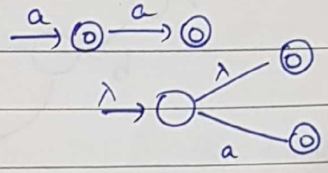
Reverse → $\{ba\}$

→ Closed under reverse.

Final becomes initial state and vice versa.

③ Prefix of L.

→ $L = \{ab, bba\}$



Prefix(L) = $\{\lambda, a, b, ab, bb, bba\}$.

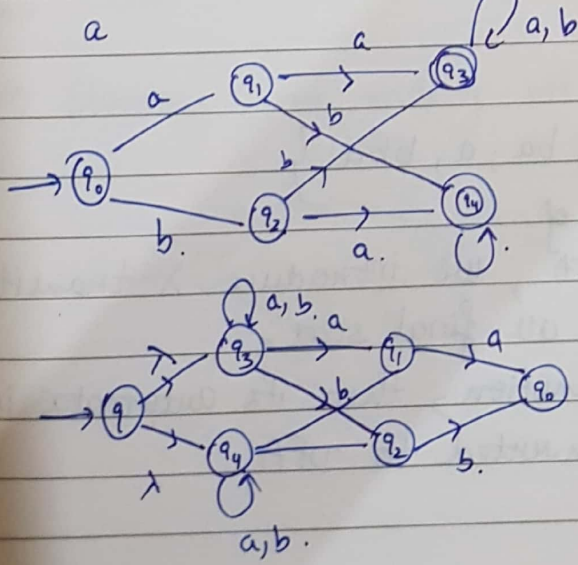
Converting non-final to final will give DFA of prefix of L.

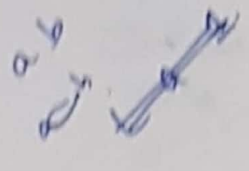
→ Closed under prefix of L.

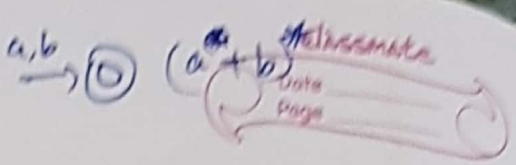
Q) Design DFA of L and prefix(L).

→

$L = \{ab, bba\}$





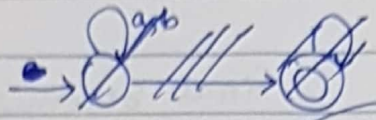


→ $L = \{ ab, bba \}$

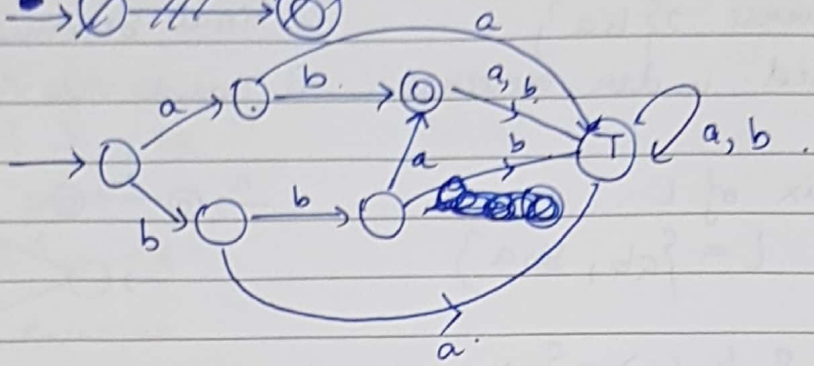
$L_p = \{ \lambda, a, ab, b, bb, bba \}$

- Non-final states are made final for drawing DFA of prefix

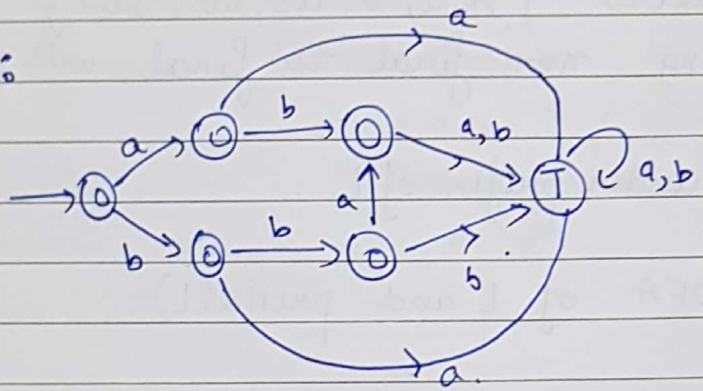
→



DFA (L):



DFA (L_p) :



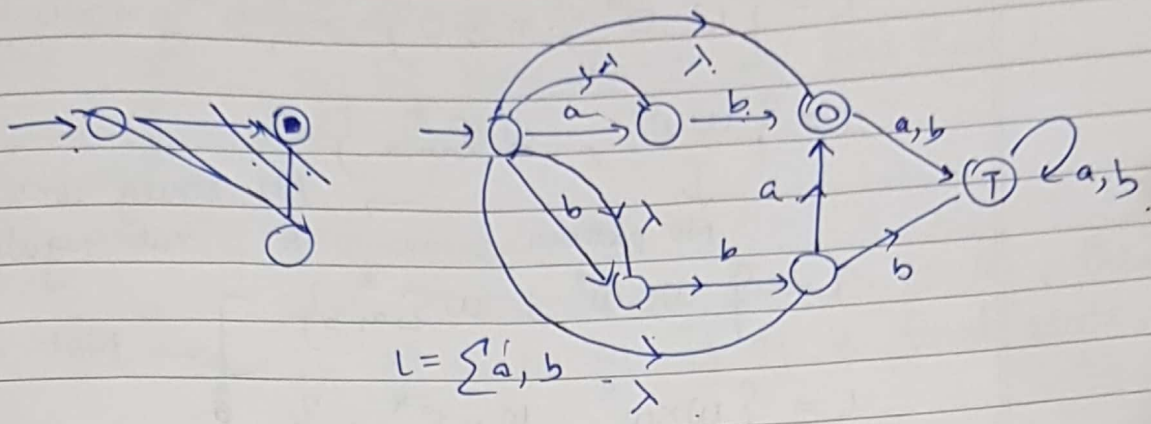
④ Suffix

$L = \{ ab, bba \}$

$L_s = \{ \lambda, ab, ab, ba, a, bba \}$

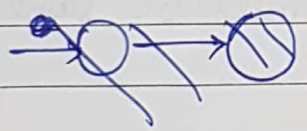
For drawing suffix of DFA, we introduce λ transition from initial state to all final state.

If there is λ transition, then it's automata is NFA. Every NFA can be converted to DFA



→ $L = \{ a^{2^i} ; i \geq 0 \}$

$L = \{ a, a^2, a^4, a^8 \ldots \}$



No DFA is possible because no pattern.

→ $L = \{ a^n b^m ; n \leq m \}$

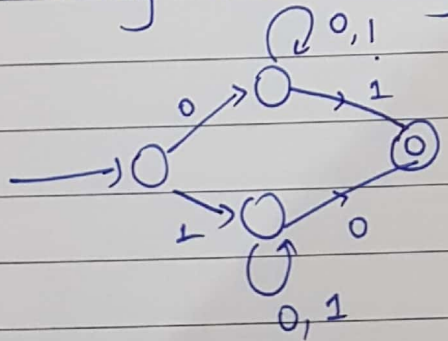
→ ~~$L = \{0, 1\}$~~

$L = \{0, 1\}$.

$L = \{$All strings with equal number of 01 and 10$\}$

Eg 0110101 is not accepted by this L.

L ⇒ Starting and ending bit is same.



Regular expression:

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

→

$R = (a+b)^*$ $\quad w = ab$

$ab(a+b)^*bb$ $\quad$ classmate Date Page $\quad$ $aba(a+b)^*abb$

→ $L = \{0^n 0^n ; n \geq 0\} = \{0^{2n} ; n \geq 0\}$ → Regular

$L = \{ww^R ; w \Sigma^*_{a,b}\}$

↓ No pattern ∴ no DFA ∴ not regular.

↳ All string consisting of a and b.

$L = \{wcw^R ; w \Sigma^*_{a,b}\}$ — Not regular.

$L = \{wxw^R , w,x \Sigma^*_{a,b}\}$

↳ This is regular.

$(a+b)^*$

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

$ab(a+b)^*ba$

↳ This can be seen as X.

It's starting and ending will be same.

→ $a^n c a^n$ → Not regular.

$a^n x a^n$ → Regular.

$a^n (a?$

Q $L = \Sigma 0,1$

↳ Accepts binary string whose decimal equivalent is divisible by 5.

→ 000, 101, 1010, 1111, ...

ab → a

a → b

classmate Date Page

- Conversion of NFA to DFA

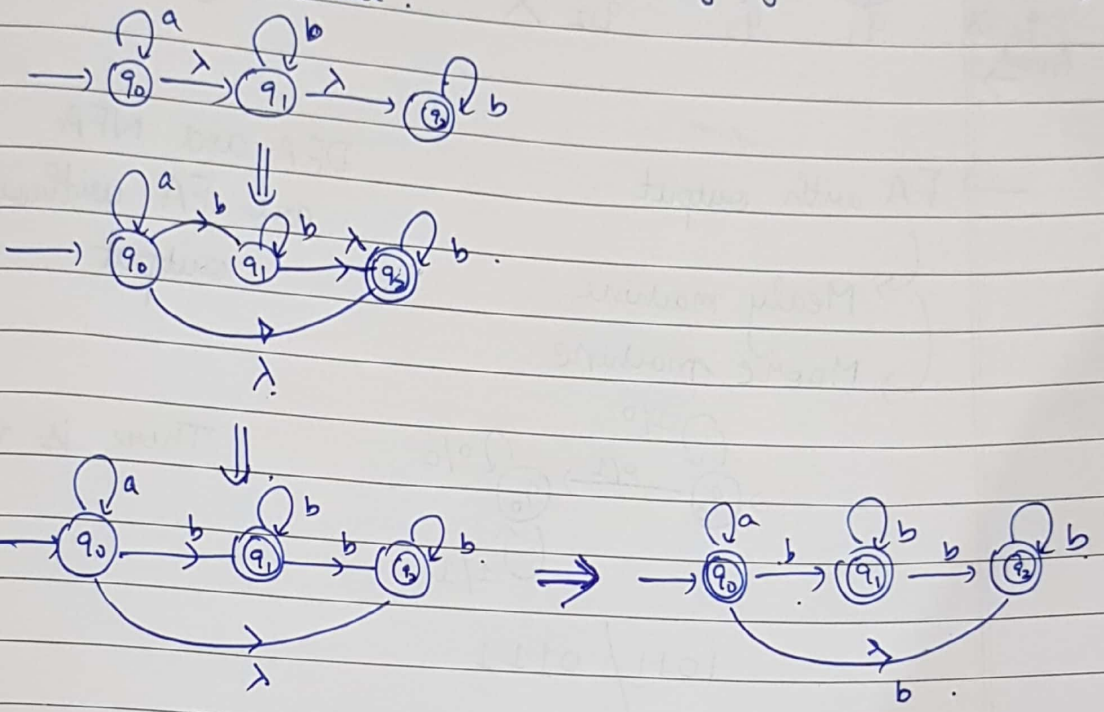
→ In NFA, there can be more than 1 or less than 1 transition for single input.

Conversion steps:

→ (1) Remove all the λ transition

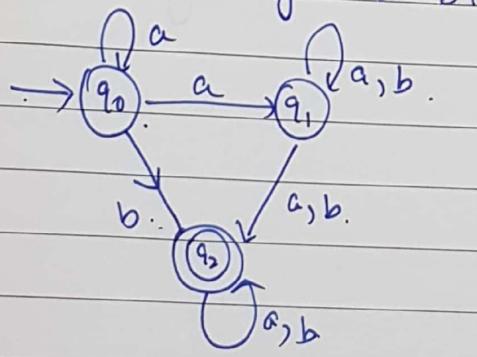
(2) → If there exists λ transition from a_i to a_j, then copy all transition of a_j to a_i. If a_j is final state, then make a_i final state.

→



(2) Make state table of DFA by NFA.

→



NFA

	a	b
q_0	$q_0 q_1$	q_2
q_1	q_2	$q_1 q_2$
(q_2)	q_2	q_2

DFA

→

	a	b
q_0	q_{01}	q_2
q_{01}	q_{012}	q_{12}
q_{012}		

From NFA, in first row, if there are more than one trans merge them.

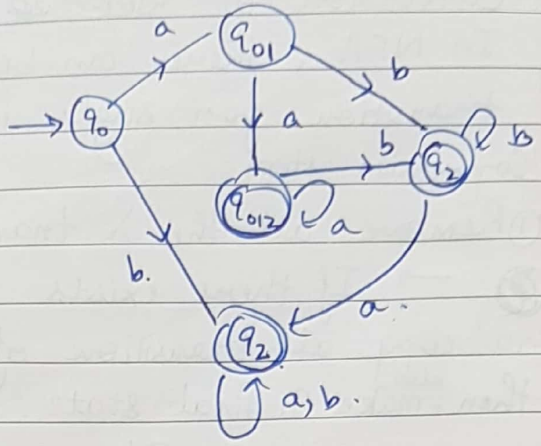
1-5:30. 5th. 7th.

DFA.

	a	b
q_0	q_{01}	q_2
q_{01}	q_{012}	q_{12}
q_{012}	q_{012}	q_{12}
q_{12}	q_2	q_{12}
q_2	q_2	q_2
q_1	q_2	q_{12} X

Final State

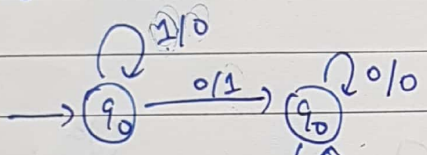
∵ q_2 is final.



→ FA with output.

- Mealy machine
- Moore machine. → Output depends ~~on~~ only on state.

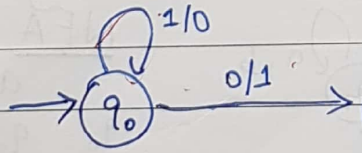
DFA and NFA are FA without output.



There is no final state

1011 / 0111

Q. Draw machine to increment binary number by 1.



110 + 1 = 001

001 + 1 = 010

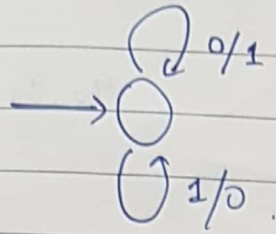
111 + 1 = 000

101 + 1 = 110

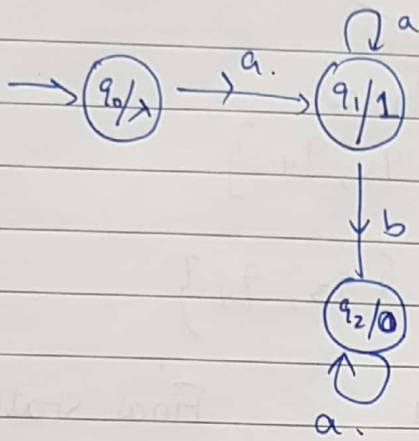
1000100 + 1 = 1

1010 + 1 = 1011

Mealy machine to complement a binary number



→ Moore machine.

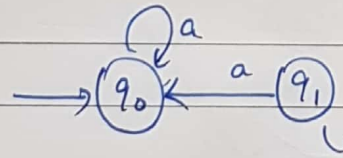


aab
120

Mealy and moore machine are equivalent in power i.e. number of strings accepted by mealy and moore machine are equal.

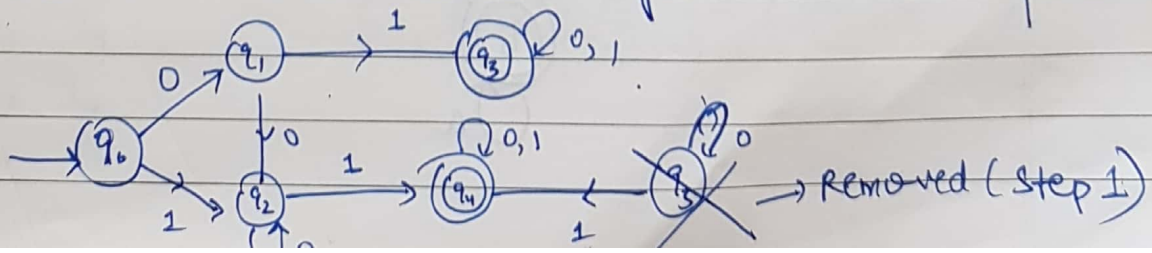
Minimization of DFA

(1) Remove all the states unreachable from initial state.



↳ q1 is not reachable from q0.

(2) Divide set of state into two subset - set of final state and sets of non-final state.
Further divide those until no further division possible



(3) Draw its transition table

	0	1
q_0	q_1	q_2
q_1	q_3	q_3
q_2	q_2	q_4
q_3	q_3	q_3
q_4	q_4	q_4

	0	1
q_0, q_1	q_1, q_2	$q_2 q_3$
	↓	↓
	This is a subset	This is not a subset

So, q_0 and q_1 can't be merged

Non-final $\{q_0, q_1, q_2\}$ Final $\{q_3, q_4\}$

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

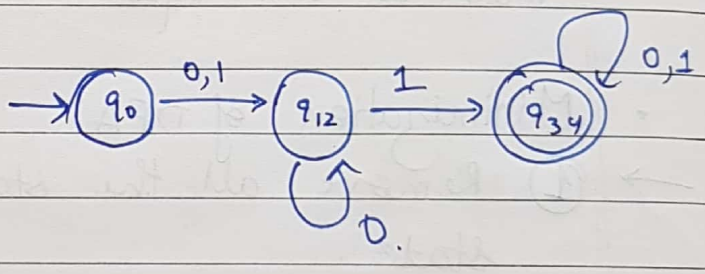
↓

Output on 1 is q_3, q_4 is in same set

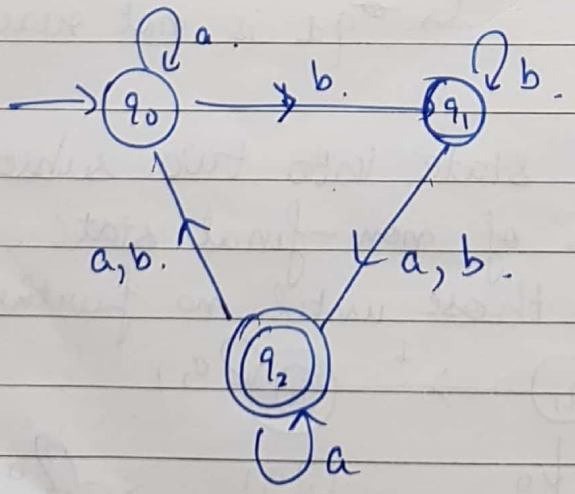
Final states become q_0, q_{12}, q_{34}.

Transition Table.

	0	1
q_0	q_{12}	q_{12}
q_{12}	q_{12}	q_{34}
q_{34}	q_{34}	q_{34}



Convert NFA to DFA.



$L = ab$ w $(a+b)^*$

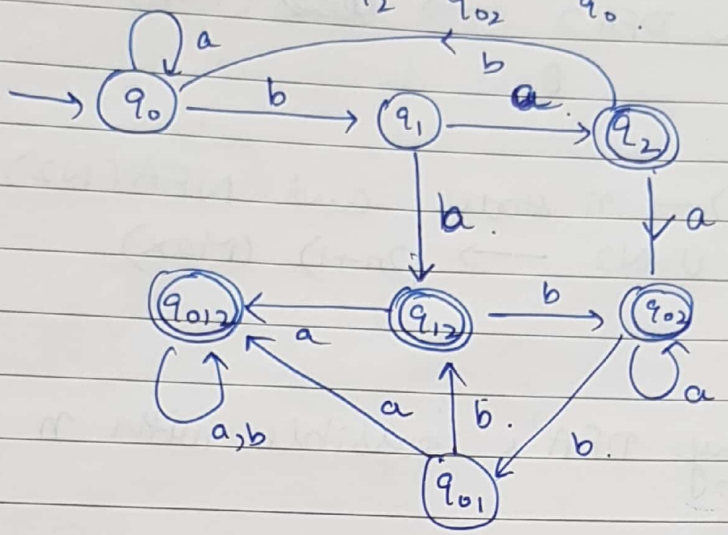
NFA

	a	b
q_0	q_0	q_1
q_1	q_2	q_{12}
(q_2)	q_2, q_0	q_0

DFA

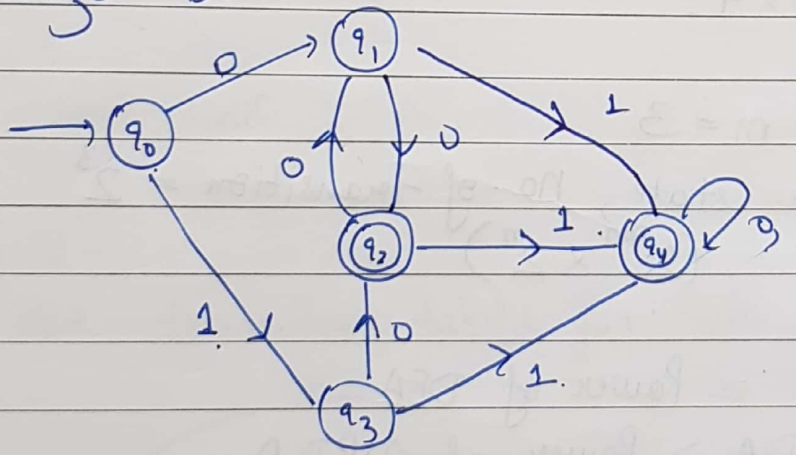
	a	b
q_0	q_0	q_1
q_1	q_{02}	q_{12}
q_{12}	q_{012}	q_{02}
q_{012}	q_{012}	q_{012}
q_{02}	q_{02}	q_{01}
q_2	q_{02}	q_0

	a	b
q_{01}	q_{012}	q_{12}



Q If NFA contains n states then how many max states in equivalent DFA?
→ 2^n

(2) Minimize DFA

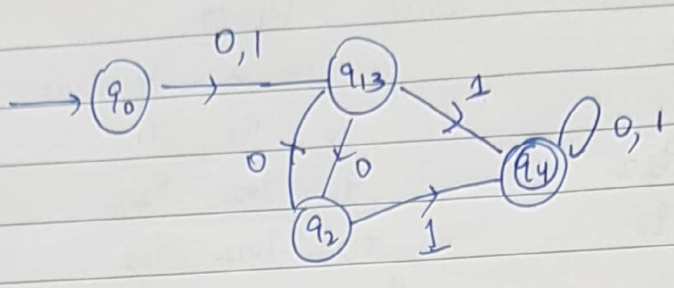


	0	1
q_0	q_1	q_3
q_1	q_2	q_4
q_2	q_1	q_4
q_3	q_2	q_4
q_4	q_4	q_4

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

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

$L = \{$



Q DFA1 → n states and DFA2 → n states.
DFA1 ∪ DFA2 → 2^n (Max)

Q NFA (N1) → n states and NFA (N2) → n states
N1 ∪ N2 → (2n+1) (Max)

Q How many DFA's possible with n states and m i/p symbol?

→ ~~n states~~

Eg ⇒ n = 2, m = 2.
~~16 × 4~~.

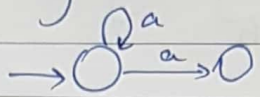
n = 2 ; m = 3
On one state, no. of transition = 2^3
$(2^{mn} \times 2^n)$

→ Power of NFA = Power of DFA
Power of NPDA > Power of DPDA
↳ Push Down

Q How to decide if 2 DFA are equivalent power?
→

$(4^m+1)^{n}$ $(n \times m)^m$

Q.2 How many NFA with 2 state and 2 i/p are possible



Possible Transition

1. Self loop
2. →
3. Self loop + →
4. Null.

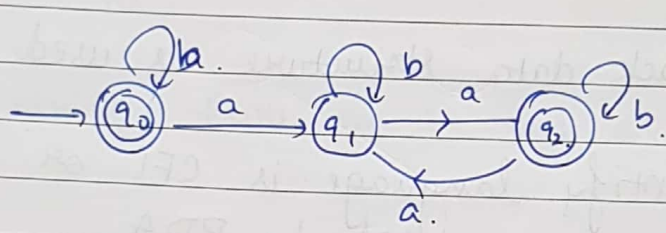
On state 1, total states = $4 \times 4 + 1 = 17$.

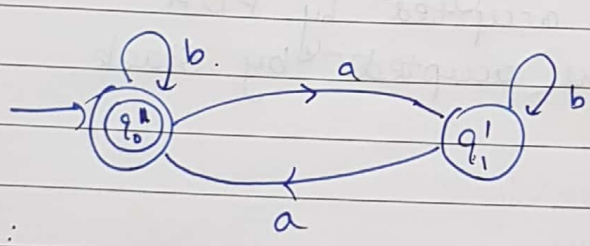
On state 2, total states = 17

4 possibilities on final state.

$(4^n+1)^m \times 2^n$

→





Way 1:

1. Minimize both DFA.

Way 2:

1. Make transition table for both DFA.

→ Chose either both final or both non-final state.

	a	b
$q_0\ q_0'$	$q_1\ q_1'$	$q_0\ q_0'$
$q_1\ q_1'$	$q_2\ q_0'$	$q_1\ q_1'$
$q_2\ q_0'$	$q_1\ q_1'$	$q_2\ q_0'$

↓

Check in all pairs if there is any pair of final and non-final, then DFA are not equivalent

Not in Midsem

Arden's Theorum → For finding regular expression corresponding to given DFA.

CONTEXT FREE LANGUAGE (Push Down Automata)
↳ PDA.

Deterministic	Non-deterministic
DCFL	NCFL.

CFL overcome disadvantage of regular language that it has no memory.

In CFL, stack data structure is used.

How to identify language is CFL or not?
1. It must be accepted by PDA
2. All languages accepted by stack.

$S \rightarrow T*$.
↓
Single non terminal.

$S \rightarrow S+S$.
$S \rightarrow S*S$.
$S \rightarrow id$.

id + id * id.

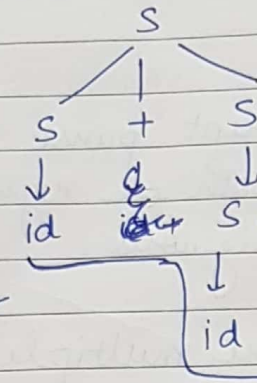
$S \rightarrow S+S$
$S \rightarrow id+S$
$S \rightarrow id+S*S$
$S \rightarrow id+id*id$.

} Sentential Form. (Intermediate forms before final form)

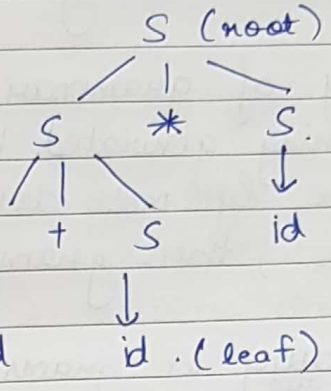
$S \rightarrow S*S$
$S \rightarrow S+S*S$
$S \rightarrow id+id*S$
$S \rightarrow id+id*id$.

→ Derivation Tree / Parse Tree.

$S \to S + S$



$S \to S * S$



We traverse from left to right and top to bottom.

→ id + id * id.

For one string, there can be one or more than one sentential form.

derivation

left most derivation

→ In this, always left most variable is replaced.

$S \to S + S$

$\to id + S$

$\to id + S * S$

$\to id + id * S$

$\to id + id * id$

rightmost derivation

Always right most is replaced.

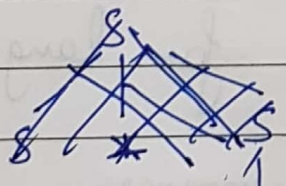
$S \to S * S$

$\to S * id$

$\to S + S * id$

$\to S + id * id$

$\to id * id * id$



There is no change in parse tree for leftmost and rightmost.

→ For a string, parse tree for left most and right most is same always.

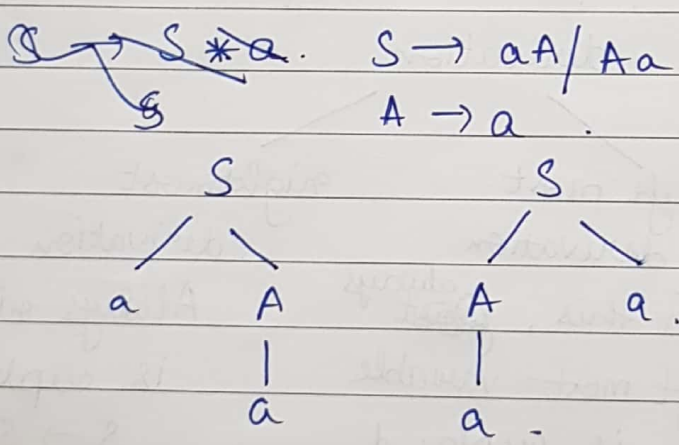
→ Ambiguity of grammar:
If a string genrated by two different parse tree using either left most derivation for both or rightmost derivation, then grammar is ambiguous.

For a string in grammar, if there are multiple left most derivation true, or multiple rightmost derivation true, then grammar is ambiguous.

→ Context Free Grammar is ambiguous

→ Regular — Two non-terminal can't come together

S → aA / Aa
A → a



So, regular grammar is ambiguous.

→ All programming language are designed with context free grammar.

We can use multiple grammar for language.

C language → Context sensitive grammer
In context free grammar there is no, Concept of forward declaration.

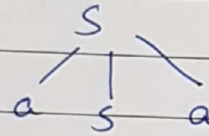
context free

Ambiguity of grammar is undecidable because there is no algorithm for.

Language cannot be ambiguous.

There exists some language for that no unambiguous grammar is available. So that language is known as inherently ambiguous language. Eg. non deterministic context free language.

① $S \to aSa \mid bSb \mid SS \mid \lambda$



$S \to aSa$.
$S \to aa$.

$S \to SS$
$S \to aSaS$
$S \to aaa$

$S \to SS$
λS
$\lambda\lambda$
$S \to \lambda$

$S \to \lambda$

$S \to aSbS \mid bSaS \mid \lambda$.

$S \to aSbS$ $S \to bSaS$ $S \to \lambda$

ab

$S \to aSbS$
$S \to aaSbbS$
$S \to aabb$

ba

$S \to aSbS$
$S \to ab\lambda$

These are diff.

$S \to aSbS$
$aSbS$
$abSaSbS$
$abab$

Same

So, ambiguous.

Regular → 5 tupple
CFL → 7 tupple.

abc

→ $L = a^n b^n$ is context free.

→ PDA is FA with a single stack with infinite length.
PDA with finite length stack → Not CFL
↳ Regular.

→ Stack → LIFO.
Example of context free.
→ $L = a^n b^{2n}$.
• We read first b and for second b, pop a.

$L = a^n b^m c^n$; $n, m \geq 0$.
Push a. Read b. When c comes, pop a.

If variables have 2 equalities, then it is not context free
$(a^n b^n c^n)$

$L = a^n b^m c^n d^m$; $n, m \geq 0$.
↳ Not possible (Not CFL)
$L = a^n b^n c^m d^m$.
↳ CFL

→ $L = wcw^R$; $w \in \Sigma^*_{a,b}$.
↳ CFL (Deterministic).

$L = ww^R$
abbbba. → Till which point, we will push we don't know.
↳ It in NCFL.

$L = a^n b^m c^k$
$n = m$ or $m = k$.
↳ NCFL
because we don't know when to push and when to pop.

No conversion in NCFL and DCFL