(C)

* Moore Machine	Mealy Machine
• Output depends only upon present state	Output depends on the present state as well as present input.
• If input change, the output does change.	If input change, output also changes
• More state is required	Less state is required
• Less hardware requirement for circuit implementation	More hardware requirement for circuit implementation.
• React slower to input	React faster to input
• Synchronous output & state generation	Asynchronous output generation
• Output is placed on states	Output is placed on transitions
• easy to design	difficult to design.

Ques Describe Regular expressions?

Ans. • It is a sequence of pattern that defines a string.
• It is a most effective way to represent any language.
• language accepted by regular expression is known as regular language.
• It is used to match character combination in string.
• ϕ is regular expression for regular language ϕ.
• ϵ is regular — " — " ——— $\{\epsilon\}$
• if a & b is regular exp^n then a+b is also reg. exp^n with lang $\{a, b\}$
• — " ——— " — ab — " ——— regular.

Ques Describe Theory of automata?

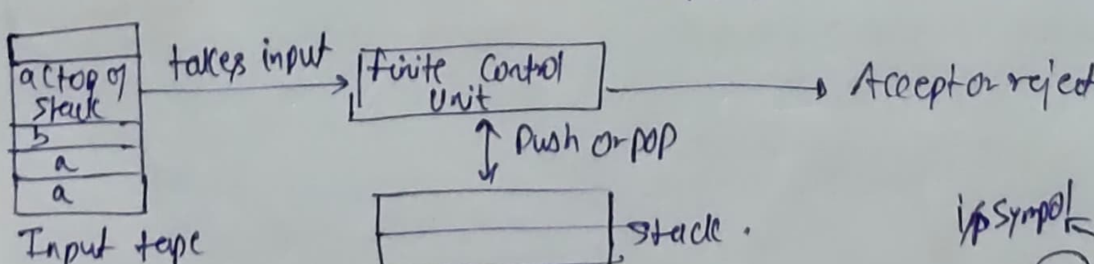
Ans. • It is a theoretical branch of computer science & Mathematics.
• It is a study of abstract machine & computation problem which can be solved using these machines.
• The abstract machine is called automata.
• Main motivation behind developing the automata theory was to develop method to describe & analyse the dynamic behaviour of discrete systems.
• Automata is a machine which takes string as input and input goes through finite no. of state and may enter in final state.
• It is consist of states & transitions.

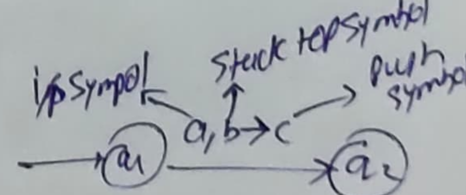
Ques Difference between NP-hard & NP-Complete.

NP-hard	NP-complete
• NP-hard problems (X) can be solved if & only if there is a NP-Complete problem (Y), that can be reducible into X in polynomial time.	NP-complete problems can be solved by Non-deterministic Algo / Turing machine in polynomial time.
- To solve problem it do not have to be in NP	To solve such problem it has to be in both NP and NP-hard.
- Do not have to be decision problem.	It is exclusively a decision problem.
- ex:- Halting problem, vertex cover problem.	ex: circuit satisfiability problem.

Ques What is Pushdown Automata?

- It is a way to implement context free grammar in a similar way to design DFA (deterministic finite automata) for a regular grammar.
- DFA can remember finite amt. of info but PDA can remember ∞ amt. of info.
- PDA follows " finite state machine + stack ".
- PDA has 3 component
  - Input tape
  - Control unit
  - A stack with ∞ size.





- It has 7 tuples: (Q, Σ, S, δ, q_0, I, F) — set of accepting state.

Q — finite no of state
Σ — i/p Alphabet
S — Stack Symbol
δ — transition function
q_0 — initial state
I — initial state top symbol

... problems can be

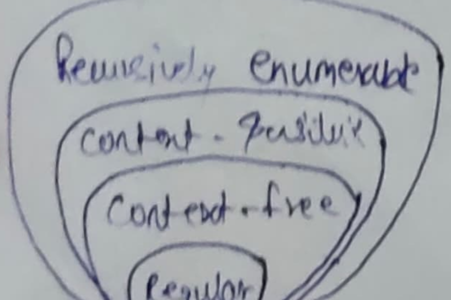
Ques Describe Universal Turing Machine ?

Ans
- Programmable TM is called UTM.
- It provides a soln to problems that are computable.
- It minimizes space complexity.
- UTM is subset of all TM.
- Transition funn is $Q \times T \rightarrow Q \times T \times \{L, R\}$ | Q is finite set of states, T is Tape of alphabet
- Although developed for theoretical reasons, It helped in development of stored program computers.

Ques Describe chomsky classification of Grammar.

Ans:: Acc to Chomsky, there are 4 types of grammars - Type-0, Type-1, Type-2, Type-3

Grammar type	Grammar Accepted	Language Accepted	Automata
Type-0	Unrestricted Grammar	Recursively enumerable	Turing Machine
Type-1	Context-sensitive	Context-sensitive	Linear-bounded
Type-2	Context-free	Context-free	Pushdown
Type-3	Regular grammar	Regular language	Finite state



Type-0 Production
- $\alpha \rightarrow \beta$
- α cannot be null
- β is a string of terminal & non-terminal
- α is a string —// —— //— with atleast one non-terminal

Type-1 Production $\alpha A \beta \rightarrow \alpha Y \beta$
- α & β may be empty
- Y must non-empty

Type-3
Production
$X \rightarrow a$ or $X \rightarrow aY$
$X, Y \in$ Non terminal.
$a \in$ Terminal.

Type-2 Production $A \rightarrow Y$
- A (non-terminal
- Y (string of terminal & non-terminal)

Ques Describe Context free grammar? (4)

Ans. • CFG is used to generate all posible strings in given formal language.

• can be described as 4 Tuples:

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

∴ G : Grammar

V : finite set of non-terminal symbol

T : —"— Terminal symbol

P : set of production rules.

S : Start symbol.

• CFG can be classified on the basis of following two properties:

(1) Based on no. of strings it generates:-

• If it generate finite no. of strings, then CFG is non-Recursive grammar.

• If —"— infinite no. of strings, then CFG is Recursive grammar.

(2) Based on no. of derivation tree:-

• If there is only 1 derivation tree, then CFG is unambigous.

• If there is more than 1 derivation tree, then CFG is ambigous.

Ques Define Turing Machine?

Ans:- • It is an accepting device which accepts the languages generated by type 0 grammars.

• (TM) is a mathematical model which consist of an ∞ length tape divided into cells on which input is given.

• Described as a 7 tuple $(Q, X, \Sigma, \delta, q_0, B, F)$

F : set of final state.

B : Blank symbol

q_0 : initial state

δ : transition function

Σ : Input Alphabet

X : tape Alphabet

Q : finite set of states

• It consist of head which reads the input

• A state register which stores the state of Turing Machine.

•

Machine	Stack Data Structure	Deterministic?
finite Automata	N.A	yes
Pushdown Automata	LIFO	No
Turing Machine	Infinite Tape	yes

• Time complexity $T(n) = O(n \log(n))$

Space —"— $S(n) = O(n)$

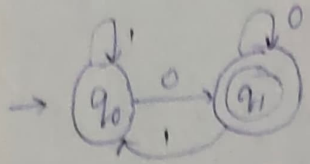
can be described as 4 tuples.

4

Ques Diff. b/w DFA & NFA

DFA	NFA
• refers to Deterministic finite Automata	• refers to Non-Deterministic Finite Automata
• Said to DFA if corresponding to an i/p symbol there is single resultant state i.e only one transition.	• Said to NFA if there is more than one possible transition from one state to same i/p symbol
• All DFA is NFA	• Not all NFA are DFA.
• DFA is more difficult to construct	• NFA is easier to construct
• Requires more space.	• Requires less space
• Backtracking is allowed in DFA	• Backtracking is not allowed at always. in NFA

→ Transition diag:



Transition function

$\delta(q_0, 0) = q_1$
$\delta(q_1, 1) = q_0$
$\delta(q_1, 0) = q_1$
$\delta(q_1, 1) = q_0$

©

Mathematical model of t.d.

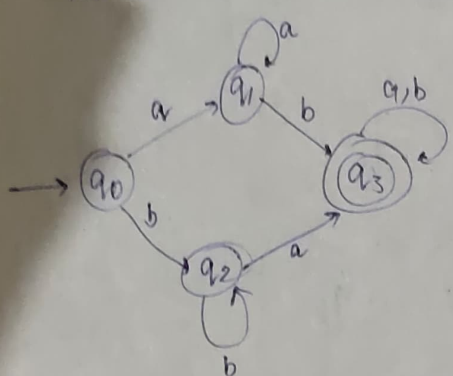
$Q = (q_0, q_1)$ set of states

Σ or i/p $= (0, 1)$

q_0 = start state

q_1 = final state

Transition dig → transit table



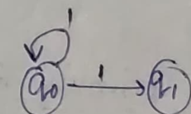
state →	a	b
$\rightarrow q_0$	q_1	q_2
q_1	q_1	q_3
q_2	q_3	q_2
q_3	q_3	q_3

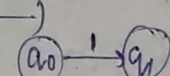
(input)

---

finite state Automata

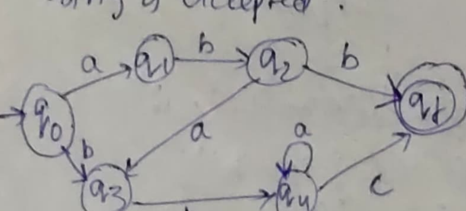
- finite no. of states
- 5 tuples
- Two types of FSA
  - NFA (non-deterministic final state Automata)
  - DFA (deterministic f, – – –, —)
- Tuples
  - Q = set of states
  - Σ = set of input alphabets
  - δ = trans^ fun^ = Q^* i/p > state
  - Initial state q_0
  - final state F





Acceptance of string by DFA

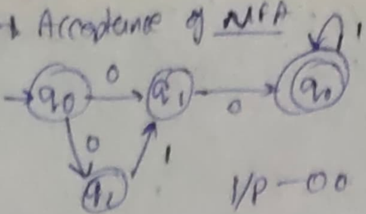
→ Rule for acceptance of string is that on traversing if machine reaches from initial state to final state then string is accepted.



String is abab aaac

$\delta(q_0, \underline{a}babaaac) \rightarrow \delta(q_1, \underline{b}abaaac) \rightarrow \delta(q_2, \underline{a}baaac) \rightarrow$
$\delta(q_3, \underline{b}aaac) \rightarrow \delta(q_4, \underline{a}aac) \rightarrow \delta(q_4, \underline{a}ac) \rightarrow$
$\rightarrow \delta(q_4, \underline{a}c) \rightarrow \delta(q_4, \underline{c}) \rightarrow \delta(q_f, \wedge)$ accepted

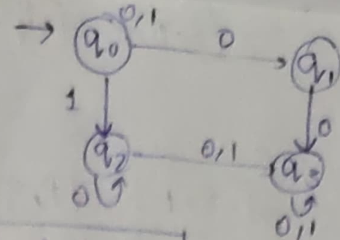
→ Acceptance of NFA



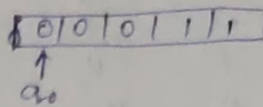
I/P – 00

$\delta(q_0, 00) \vdash \delta(q_1, 0) \vdash \delta(q_0)$ → final state reached

$\delta(q_0, 00) \vdash \delta(q_2, 0) \vdash$ X not reached.



I/P 00011



$\delta(q_0, 00011) \vdash \delta(q_1, 0011) \vdash \delta(q_3, 011)$
$\vdash \delta(q_3, 11) \vdash \delta(q_3, 1) \vdash \delta(q_3 \text{null})$
– Accepted

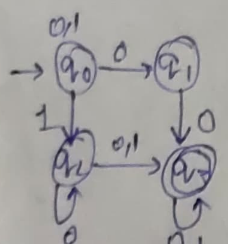
→ Conversion of NFA to DFA

(H)

→ for every NFA we have a Equivalent DFA
→ equivalent DFA, → accept same type of string as accepted by NFA
→ For conversion of NFA to DFA two different state which are behaving in similar manner will be combined into single state and new state will be born, and the behavior of new state will be similar to both state.

$(P, r) \to q$

$\boxed{\delta(P,0) \cup \delta(r,0) \approx \delta(q,0)}$



Step-1 (Make transition table for given NFA)

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

Step-2 (Group similar state / Make TT of DFA)

	0	1
q_0	$\{q_0, q_1\} = P$	$(q_0, q_2) = r$
P	$\delta(P,0)$	$\delta(P,1)$
r	$\delta(r,0)$	$\delta(r,1)$

$\delta(P,0) = \delta(q_0,0) \cup \delta(q_1,0)$
$(q_0, q_1) \cup q_3$
(P, q_3)

$\delta(P,1) = \delta(q_0,1) \cup (q_1,1)$
$= r, \phi$
$= r$

	0	1
q_0	P	r
P	$(P, q_3) = s$	r
r	$(P, q_2 q_3) = t$	$(r, q_3) = u$

$\delta(r,0) = \delta(q_0,0) \cup (q_2,0)$
$= P \cup (q_2, q_3)$
$= (P, q_2 q_3) = t$

$\delta(r,1) = \delta(q_0,1) \cup (q_2,1)$
$= r \cup q_3$
$= r \cup q_3$

	0	1
$\to q_0$	p	r
p	s	r
r	t	u
(s)	s	u
(t)	$v = q_0q_1q_2q_3$	u
(u)	v	u
v	v	u

$\delta(s,0) = (p, q_3) \to \delta(p,0) \cup \delta(q_3, 0)$
$p \cup q_3 \cup q_3$
(p, q_3)

$\delta(s,1) = (p, q_3) \to \delta(p,1) \cup \delta(q_3, 1)$
$r \cup q_3 = u$

$\delta(t,0) = (p, q_2, q_3) \to \delta(p,0) \cup \delta(q_2,0) \cup \delta(q_3,0)$
$\to (q_0, 0) \cup (q_1, 0) \cup (q_2, 0) \cup (q_3, 0)$
$\to q_0, q_1, q_2, q_3 = v$

$\delta(t,1) = (p, q_2, q_3) \to \delta(p,1) \cup (q_2,1) \cup (q_3,1)$
$\to \delta(q_0,1) \cup (q_1,1) \cup (q_2,1)$
$\cup (q_3,1)$

$(q_0, q_2) \cup \phi \cup q_3 \cup q_3$
q_0, q_2, q_3

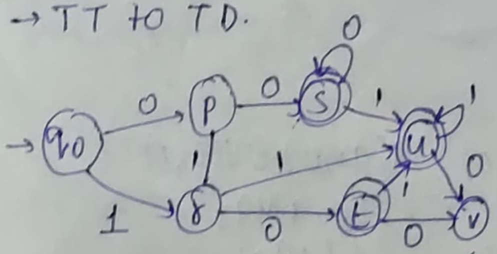
$\delta(v,0)$
$= (q_0q_1q_2q_3 \cup 0)$
$(q_0, q_1) \cup (q_3) \cup (q_2q_3)$
$\cup q_3)$

$\delta(v,1)$
$= q_0q_1q_2q_3 \cup 1$
$q_0q_2 \cup \phi\, q_3 \cup q_3$
$= u$

$\delta(u,0) = (q_0, q_2, q_3, 0)$ ①
$= \delta(q_0,0) \cup (q_2,0) \cup (q_3,0)$
$q_0, q_1 \cup q_2, q_3 \cup q_3$
$= v$

$\delta(u,1) = (q_0q_2q_3 \cup 1)$
$= \delta(q_0,1) \cup \delta(q_2,1) \cup (q_3,1)$
$= q_0, q_2 \cup q_3 \cup q_3$
$= q_0q_2q_3$
$= u$

→ TT to TD.



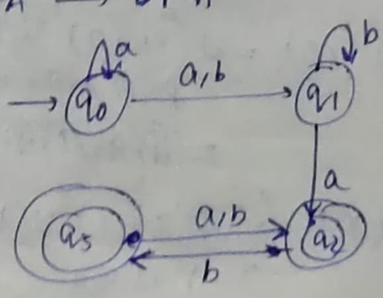
~~000~~ 000110 → check string acceptn.

TT of NFA

	a	b
$\to q_0$	q_0, q_1	q_1
q_1	q_2	q_1
(q_2)	-	q_3
(q_3)	q_2	q_2
q_0q_1	$q_0q_1q_2$	q_1
($q_0q_1q_2$)	$q_0q_1q_2$	q_1q_3
(q_1q_3)	q_2	q_1q_2
(q_1q_2)	q_2	q_1q_3

Ques NFA ⟶ DFA

Q



$q_0q_1 = p$
$q_0q_1q_2 = r$
$q_1q_3 = u$ $\quad q_1q_2 = t$

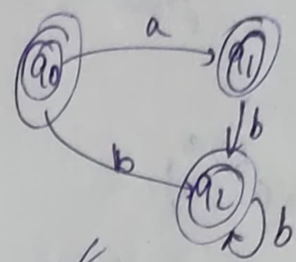
$\delta'(q_2, b) = \epsilon\text{-clos}(\delta(\epsilon\text{-clo}(q_2), b))$
$= \epsilon\text{-clos}(\delta(q_2, b))$
$= \epsilon\text{-clo}(q_2)$

$\boxed{\delta'(q_2, b) = q_2}$

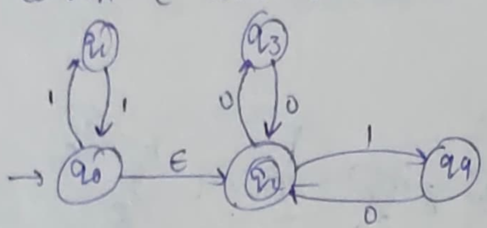
(4)

TT after removal of ϵ moves.

	a	b
$\rightarrow q_0$	$q_1 q_2$	$\emptyset$
q_1	$\emptyset$	q_2
q_2	$\cancel{q_2}$	q_2



Q convert ϵ-NFA to NFA



ϵ-closure of $(q_0) = \{q_0, q_2\}$
ϵ-clo $(q_1) = \{q_1\}$
ϵ-clo $(q_2) = \{q_2\}$
ϵ-clo $(q_3) = \{q_3\}$
ϵ-clo $(q_4) = \{q_4\}$

$\boxed{\delta'(q, a) = \epsilon\text{-closure}(\delta(\epsilon\text{closure}(q), a))}$

$\delta'(q_0, 0) = \epsilon\text{-clos}(\delta(\epsilon\text{ closure } q_0), 0))$
$= \epsilon\text{-clos}\delta((q_0, q_2), 0))$
$= \epsilon\text{-clo}(\delta(q_0, 0) \cup \delta(q_2, 0))$
$= \epsilon\text{-clo}(\emptyset \cup q_3)$
$= \epsilon\text{-clos}(q_3)$

$\boxed{\delta'(q_0, 0) = q_3}$

$\delta'(q_1, 1) = \epsilon\text{-closu}(\delta(\epsilon\text{clos}(q_1), 1))$
$= \epsilon\text{-closure}\delta(q_1, 1))$
$= \epsilon\text{-clos } q_0$

$\boxed{\delta'(q_1, 1) = q_0, q_2}$

$\delta'(q_0, 1) = \epsilon\text{-closure}(\delta(\epsilon\text{closure}(q_0), 1))$
$= \epsilon\text{-clo}(\delta(q_0, q_2), 1)$
$= \epsilon\text{-clo}(\delta(q_0, 1) \cup \delta(q_2, 1))$
$= \epsilon\text{-clo}(q_1 \cup q_4)$

$\boxed{\delta'(q_0, 1) = q_1 q_4}$

$\delta'(q_1, 0) = \epsilon\text{-closure}(\delta(\epsilon\text{closure}(q_1)), 0))$
$= \epsilon\text{-closure } \delta(q_1, 0))$
$= \emptyset, \emptyset$

$\boxed{\delta'(q_1, 0) = \emptyset}$

$\delta'(q_2, 0) = \epsilon\text{-clos}(\delta(\epsilon\text{closure}(q_2)), 0))$

$\boxed{\delta'(q_2, 0) = q_3}$

$\delta'(q_2, 1) = \epsilon\text{-clos}(\delta(\epsilon\text{-clos}(q_2)), 1))$
$= \epsilon\text{-closure}(\delta(q_2, 1))$

$\boxed{\delta'(q_2, 1) = q_4}$

$\delta'(q_3, 0) = \epsilon\text{-clos}(\delta(\epsilon\text{-clo}(q_3)), 0))$
$= \epsilon\text{-clo}(\delta(q_3, 0))$

$\boxed{\delta'(q_3, 0) = q_2}$

$\delta'(q_3, 1) = \epsilon\text{-closu}(\delta(\epsilon\text{-closure}(q_3), 1)$

$\boxed{\delta'(q_3, 1) = \emptyset}$

$\delta'(q_4, 0) = \epsilon\text{-clos}(\delta(\epsilon\text{-clos}(q_4)), 0)$

$\boxed{\delta'(q_4, 0) = q_2}$

$\delta'(q_4, 1) = \epsilon\text{-clo}(\delta(\epsilon\text{-closure}(q_4)), 1)$

$\boxed{\delta'(q_4, 1) = \cancel{q_2}}$

(read only)

	0	1
→q_0	q_3	$q_1q_4 = q_A$
q_1	$\emptyset$	$q_0q_2 = q_B$
q_2	q_3	q_4
q_3	q_2	$\emptyset$
q_4	q_2	$\cancel{\emptyset_2}$
q_A	q_2	q_B
q_B	q_3	q_A

→

	0	1
→q_0	q_3	q_A
q_1	$\emptyset$	q_B
q_2	q_3	q_4
q_3	q_2	$\emptyset$
q_4	q_2	$\emptyset$
q_A	q_2	q_B
q_B	q_3	q_A

9

→ Regular Expression

re → ① special symbols → +, ', *, ( ), ε, ∅

$re_1 + re_2 = re$
$re_1 \cup re_2 = re$
$re_1 . re_2 = re$
$re(re_1 + re_2) = re.re_1 + re.re_2$

Ex

$\{\epsilon, a, aa, aaa, aaaa, \dots\}$

ε, $\boxed{a^n, n \geq 0}$

here null is allowed

To write this mathematical expression in terms of re.

re → $\boxed{a^*}$ where * - Kleene closure
* - zero or more times.

$\{a, aa, aaa, aaaa, \dots\}$

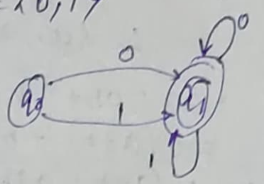
here null is not allowed

$\boxed{a^n, n \geq 1}$

$\boxed{a^+}$ → positive closure.

a is will come one or more time

Ques write a regular expression made of lang (0,1).
$\Sigma = \{0, 1\}$



$re = (0+1)^*$

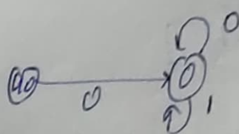
or → + (parallel path)

ε, 0, 1, 00, 101, ...

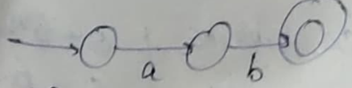
Q $\Sigma = \{0,1\}$ Begin with zero
Restriction

(0, 00, 01, 010, 01111, 0000, ...)

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



→ a.b → ab
(meaning b followed by a)



re = ab

$(ab)^* = (ab, abab, ababab, \dots)$

ⓒ

input sym.

context free grammar (CFG)

It is a formal grammar which is used to generate all possible patterns of strings in a given formal language. CFG G can be defined by four tuples as.

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

G:. set of production rules, used to generate the string of a language

V:. final set of (Variables) (non-terminal symbol). (denoted by capital letter)

T:. —//—— of termal (symbol. (—//— lower case)

S:. Start Variable.

$(V \cap T = \phi)$

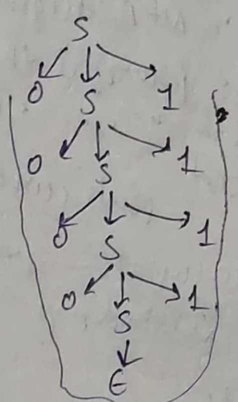
$S \to 0S1 \mid \epsilon$

$V = \{S\}$

Start $= \{S\}$

$T = \{0, 1, \epsilon\}$

$0S1 \downarrow 0S1 \downarrow 0S1 \downarrow 0S1 \downarrow \epsilon$ ⇒



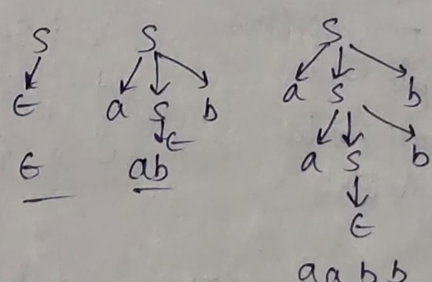
0000 1111 $\boxed{\text{String}}$

$\boxed{\text{langu}^n = \{0^n 1^n \quad n \geq 0\}}$

→ CFL → CFG

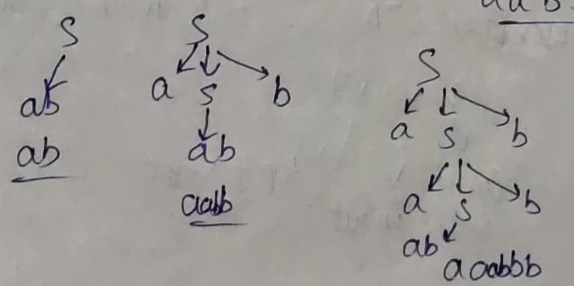
(i) $a^n b^n, n \geq 0$

$S \to aSb \mid \epsilon$



(ii) $a^n b^n, n \geq 1$

$S \to aSb \mid ab$

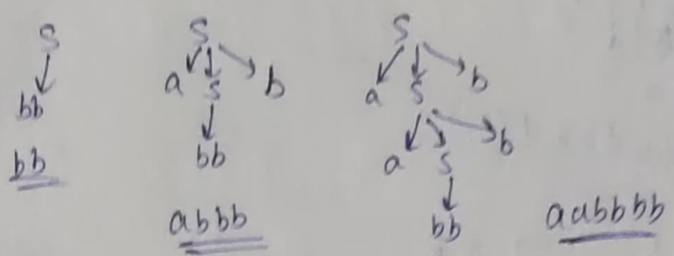


(iii) $a^n b^{n+2}, n \geq 0$

$n=0$

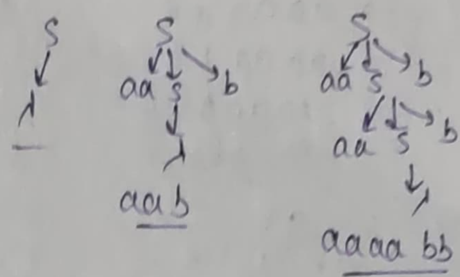
bb

$S \to aSb \mid bb$



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

$S \to aaSb \mid \lambda$

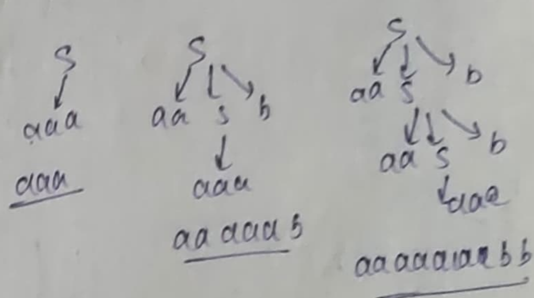


(v) $a^{2n+3} b^n, n \geq 0$

$n=0$

aaa

$S \to aaSb \mid aaa$



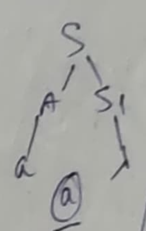
(vi) $a^m b^n, m > n$

$n \geq 0$

$S_1 \to a S_1 b \mid \lambda$

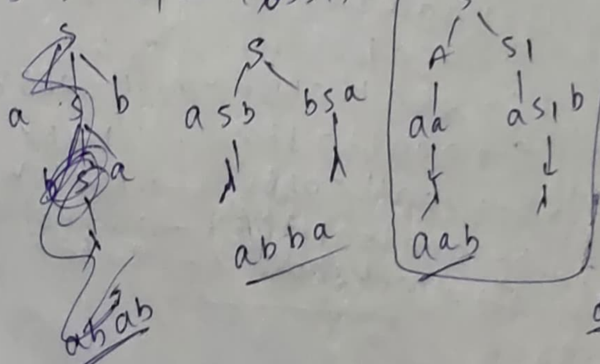
$A \to aA \mid a$

$S \to AS_1$



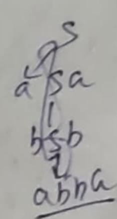
Q $\{w \mid n_a(w) = n_b(w)\}$

$S \to aSb \mid bSa \mid SS \mid \lambda$



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

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



Q $a^m b^m c^n, m, n \geq 0$

$S \to aS_1b \mid \lambda$

$C \to cC \mid \lambda$

$S \to S_1C$

Q Derivation

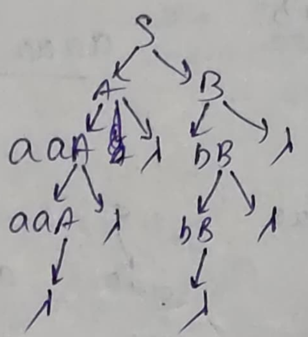
(1)

$S \to AB$
$A \to aaA \mid \lambda$
$B \to bB \mid \lambda$

check whether
$w = aaaabb \in L(G)$

$V = \{A, B\}$
$S = \{S\}$
$T = \{a, b\}$

$S \to AB$
$S \to aaAB$ (using $A \to aaA$)
$S \to aaaaAB$ (using $A \to aaA$)
$S \to aaaabB$ (using $A \to \lambda$)
$S \to aaaabbB$ (using $B \to bB$)
$S \to aaaabb$ (Hence string is accepted)



aa aa bb

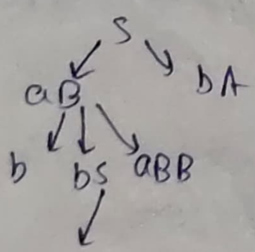
Ques

$\cancel{S \to AB}$
$\cancel{A \to aaA}$.

$S \to aB \mid bA$
$A \to a \mid aS \mid bAA$
$B \to b \mid bS \mid aBB$

string = aaabbabbba

variable $V = \{A, B\}$
$T = \{a, b\}$
$S = \{S\}$

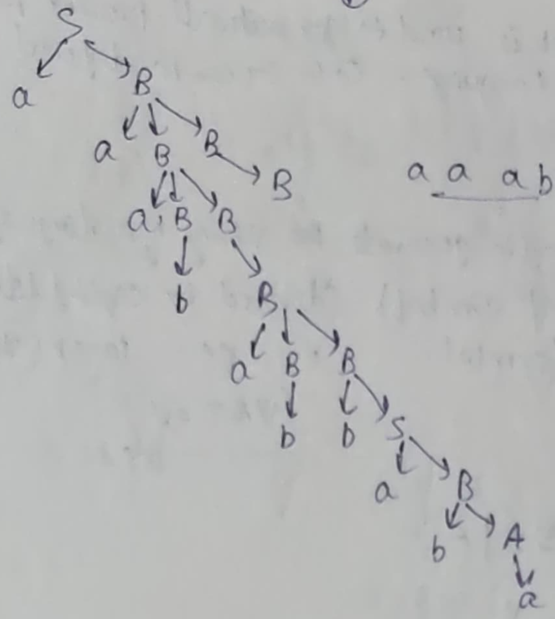


$S \to aB$
$S \to aaBB$ {using $B \to aBB$}
$S \to aaaBBB$ {—"— —"—}
$S \to aaabBB$ {—"— $B \to b$}
$S \to aaabbB$ {—"—}
$S \to aaabbaBB$ {—"— $B \to aBB$}
$S \to aaabbabB$ {—"— $B \to b$}
$S \to aaabbabbS$ {— $B \to bS$}
$S \to aaabbabbbA$ {— $S \to aB$}
$S \to aaabbabbba$ {— $S \to bA$}.
$S \to aaabbabbba$

input symbob $a, z_0 / a z_0$

derivation tree

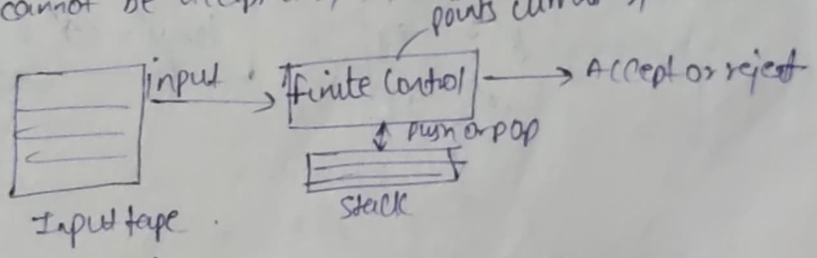
(R)



a a ababbaba

→ Push Down Automata

PDA is a way to implemed a CFG. PDA can remember infinite amount of info. It is simply an NFA augmented with an external stack memory. PDA is more powerful than FA. PDA is more superior than FA, it can accept a class of language which even cannot be accepted by FA



PDA is collection of 7 tuple. $(Q, \Sigma, \Gamma, q_0, Z, F, \delta)$

- Q: finite set of states
- Σ: input set
- Γ: stack symbol
- q_0: initial state
- Z: start symbol
- F: final set state
- δ: transition function.

Instantaneous Description (ID): tells instant descrip of PDA

(q, w, α)

- q → current state
- w → input
- α → top of stack

(A)

→ acceptance of string in PDA.

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

input string w = aaabbb

$\delta(q_0, a, z_0) = (q_0, az_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, \epsilon)$
$\delta(q_1, b, a) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, z_0) = (q_2, z_0)$

Sol^n

$\delta(q_0, \underline{a}aabbb, \underline{z_0}) \vdash q_0, \underline{a}abbb, \underline{a}z_0$ (push)
$\vdash q_0, \underline{a}bbb, \underline{a}az_0$ (push)
$\vdash q_0, \underline{b}bb, \underline{a}aaz_0$ (push)
$\vdash q_1, \underline{b}b, \underline{a}az_0$ (POP)
$\vdash q_1, \underline{b}, \underline{a}z_0$ (pop)
$\vdash q_1, \epsilon, z_0$ skip
$\vdash q_2, z_0$ string is accepted.

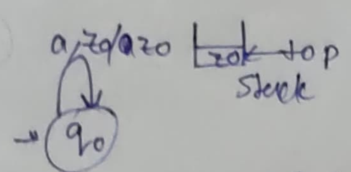
Q Construct PDA $L = \{a^n b^n \mid n \geq 1\}$

sol^n $L = \{ab, aabb, aaabbb, \ldots\}$

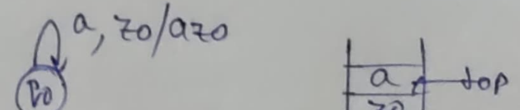
let string be aabb

step-1

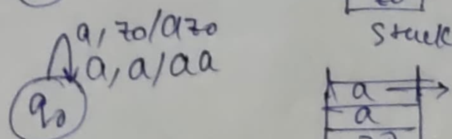
w = aabb



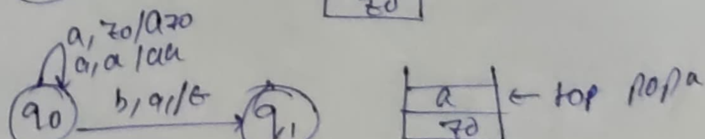
Step-2
input symbol a →



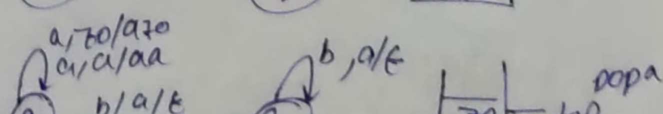
Step-3
input symb a →



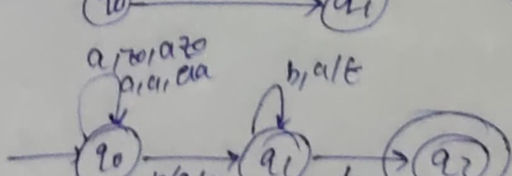
step 4 input symbol b →



step 5. input symbol b



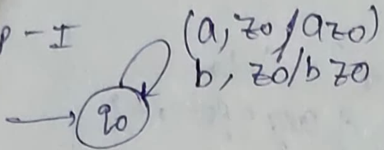
step-6 input end string



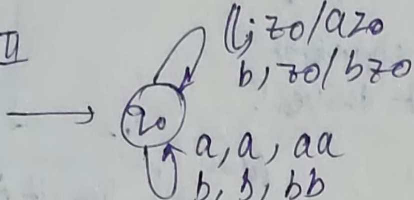
Q construct PDA $L = \{ w \mid n_a(w) = n_b(w) \}$ (9)

$\{ ab, ba, aabb, bbaa, baba \cdots \}$

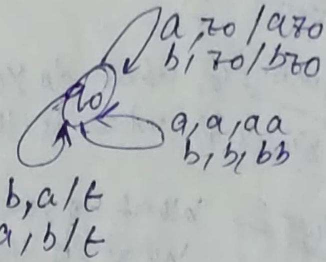
Step - I



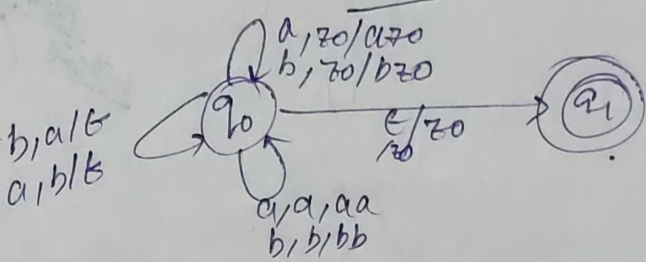
Step - II



Step - 3



Step - 4



transition fun.

$\delta(q_0, a, z_0) = (q_0, az_0)$ push
$\delta(q_0, b, z_0) = (q_0, bz_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, b) = (q_0, bb)$

$\delta(q_0, a, b) = (q_1, \epsilon)$ pop
$\delta(q_0, b, a) = (q_1, \epsilon)$

$\delta(q_0, \epsilon, z_0) = (q_1, z_0)$ or stop accepted. final state.