SYNTAX ANALYSIS

* Context free Grammars

$CFG = \{S, P, T, N\}$

S (start symbol) → symbol from which the production starts

P (Productions) → Specify the manner in which terminals & non terminals can be combined to form strings

T (Terminals) → basic symbols which forms string

N (Non Terminals) → Syntatic Variables

Example

$S \rightarrow ab \mid bA$

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

S → Start symbol

T → a, b

N → S, A

P → S→ab, S→bA, A→a, A→aS, A→bAA

* Derivations

Leftmost

Rightmost

Example $E \to E+E \mid E*E \mid -E \mid (E) \mid id$

derive $-(id+id)$

$E \to -E$
$\to -(E)$
$\to -(E+E)$
$\to -(id+E)$
$\to -(id+id)$

Left variable is replaced first

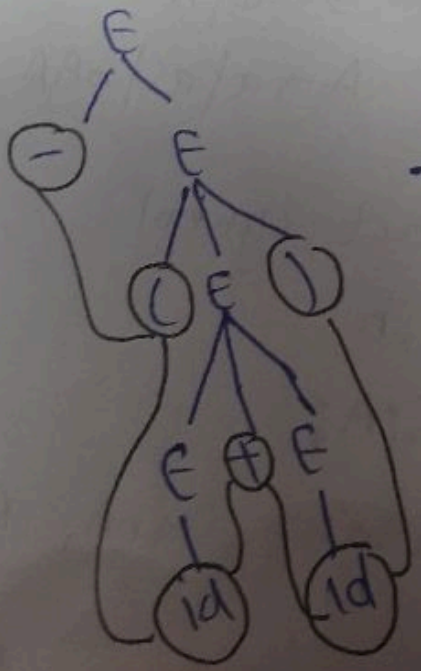
LEFTMOST DERIVATION

$E \to -E$
$\to -(E)$
$\to -(E+E)$
$\to -(E+id)$
$\to -(id+id)$

Right variable is replaced first

RIGHTMOST DERIVATION

Parse Tree



$-(id+id)$

Ambiguous Gr

Grammar

Parse tree.

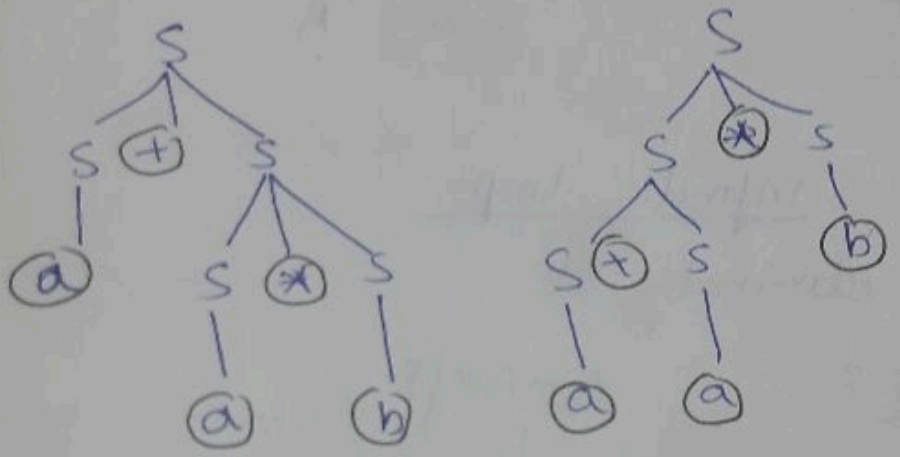
Example

(2) Ambiguos Grammar

Grammar which conclucts more than one pause tree.

Example $S \rightarrow S+S \mid S*S \mid (S) \mid a \mid b$

done $a+a*b$



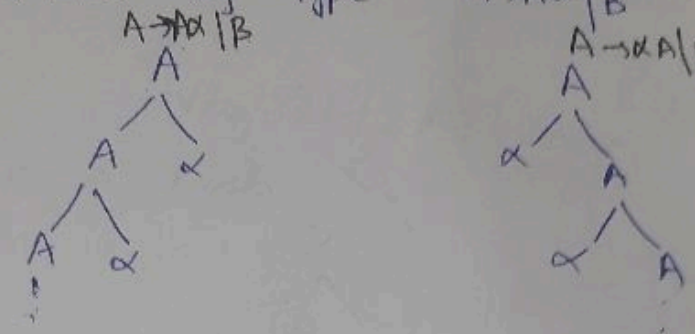
$a+a*b$
Rightmost

$a+a*b$
Leftmost

⇒ 2 different Pause Trees for same grammar.
So it is AMBIGUOS GRAMMAR

* Left Recursion

Grammar of type $A \to A\alpha \mid B$ or $A \to \alpha A \mid \beta$ ⇒ Problem



This is in infinite loop

We need to eliminate it.

JUST DO IT :

$A \to A\alpha \mid B$

↓ replace

$A \to \beta A'$
$A' \to \alpha A' \mid \varepsilon$ ⇒ Removed

EXAMPLE

$E \to E+T \mid T$
$T \to T*f \mid F$
$f \to (E) \mid id$

$A \quad A(\alpha) \quad \beta$
$E \to E+T \mid T$

$A \quad A(\alpha) \quad \beta$
$T \to T*F \mid F$

$A \to \beta A'$ $\quad E \to TE'$ $\quad T \to FT'$

$A' \to \alpha A' \mid \varepsilon$ $\quad E' \to +TE' \mid \varepsilon$ $\quad T' \to *FT' \mid \varepsilon$

∴ Grammar after removing left Recursion =

$E \to TE'$
$E' \to +TE' \mid \varepsilon$
$T \to FT'$
$T' \to *FT' \mid \varepsilon$
$f \to (E) \mid id$

$A \to \alpha A \mid \beta$

Left Factoring

→ Process of factoring out the common prefixes.

EXAMPLE

$S \to iEtSeS \mid iEtS \mid a$

$E \to b$

It becomes

$S \to iEtSS' \mid a$

$S' \to eS \mid E$

$E \to b$

Parsing Types

- TOP DOWN
  - ↳ Recursive Descent Parser
  - ↳ LL(1) Grammar
- BOTTOM UP
  - ↳ SLR
  - ↳ LR(1)
  - ↳ LALR(1)

st & Follow

es for finding first ~~of a Non Terminals~~ is -

f X is terminal, then FIRST(X) = {X}

f X is a non terminal, & X → aα, the add a to first(X).

f X → ε the add ε to FIRST(X)

f X → $Y_1 Y_2 \cdots Y_k$ & $Y_1 Y_2 \cdots Y_k$ → non terminals
first(X) = first(Y_1)

Example 1.

S → aABb
A → c | ε
B → d | ε

* To find first of the non terminals add the first non terminals which we get.

First (S) = {a}
First (A) = {c, ε}
First (B) = {d, ε}

Example 2.

S → iftSS' | a
S' → eS | ε
E → b

First S = {i, a}
First S' = {e, ε}
First E = {b}

Example3

$E \rightarrow TE$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$f \rightarrow (E) \mid id$

First(E) = FIRST(T) = FIRST(F) = { (, id }

- First(E) = { (, id }
- first (T) = { (, id }
- First (F) = { (, id }

- first (E') = { +, ε }
- first (T') = { *, ε }

Example 4

$S \rightarrow aAB \mid bA \mid \varepsilon$
$A \rightarrow aAb \mid \varepsilon$
$B \rightarrow bB \mid c$

first S = { a, b, ε }
first A = { a, ε }
first B = { b, c }.

(7) Rules for finding follow

To find follow:-

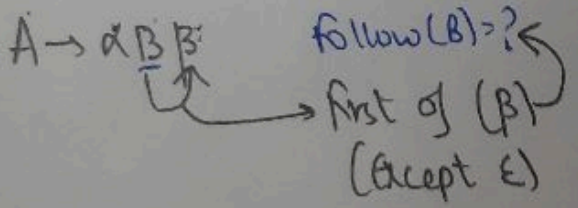
Add $ to follow(X) where X is the starting symbol.

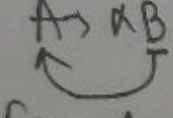
If there is a production $A \to \alpha B\beta$ where $\beta \neq \varepsilon$, then first(β) - ε is added to follow(B).

If there is a production $A \to \alpha B$ or $A \to \alpha B\beta$ where $\beta = \varepsilon$; then follow(A) is added to follow(B)

Lets Elaborate:-

1. $ is added to follow of starting symbol
2. The non terminal whose follow is to be found, we have to find it in the right side of productions.
3. Wherever the non terminal is found check the immediate next
   - if there is something → find first & add.
   - if there is nothing → then the follow of leftmost symbol is added to follow of non terminal to be found





follow A → added to follow(B)

Example

$S \rightarrow aABb$
$A \rightarrow c \mid \varepsilon$
$B \rightarrow d \mid \varepsilon$

find follow of each Non terminal.

Follow $S = \{\$\}$
FOLLOW $A = \{d, \$\}$
FOLLOW $B = \{b\}$

Example

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$F \rightarrow (E) \mid id$

FOLLOW $(E) = \{\$,)\}$
Follow $(E') = \{\$,)\}$
FOLLOW $(T) = \{+, \$,)\}$
FOLLOW $(T') = \{+, \$,)\}$
FOLLOW $(F) = \{*, +, \$,)\}$

Example

$S \rightarrow iEtSS' \mid a$
$S' \rightarrow eS \mid \varepsilon$
$E \rightarrow b$

Follow $(S) = \{\$, e\}$
Follow $(S') = \{\$, e\}$
Follow $(E) = \{t\}$

LL(1) Grammar

L → Left to right Scanning of Input
L → Leftmost derivation
1 → each step will consider one symbol of lookahead

STEPS

Convert the grammar into unambiguous. (Perform left factoring)
Remove left recursion
find first & follow
Construct Table.

While constructing the table. write non terminals on one side & terminals on other. (including $)

Lets understand with example:-

Example

$S \to A$

$A \to ab \mid Ad$

$B \to bBc \mid f$

$C \to g$

Eliminate Left Recursion

$A \to abA'$

$A' \to dA' \mid \varepsilon$

	First	follow
S	a	$
A	a	$
B	b, f	c
C	g	c $
A'	d, ε	$

Predictive parsing table

	a	b	c	d	f	g	$
S	S→A						
A	A→abA'						
B		B→bBc			B→f		
C						C→g	
A'				A'→dA'			A'→ε

* fill the production in the first of leftmost symbol
* In case $A' \to \varepsilon$ (fill this production in follow of A')

⇒ Only one entry in each cell. ∴ LL(1) Gramma

Example

$E \to iAcE \mid iAcEeE \mid a$

$A \to b$

→ Remove left factoring

$E \to iAcEE' \mid a$

$E' \to eE \mid \varepsilon$

$A \to b$

	First	Follow
E	i, a	$, e
E'	e, ε	$, e
A	b	c

LL(1) Parsing Table

	a	b	e	i	c	$
E	E→a			E→iAcEE'		
E'			E'→eE E'→ε			E'→ε
A		A→b				

→ Multiple entries

* Not LL(1) Grammar

CHOMSKY HIERARCHY

On the basis of type of productions Chomsky Classify grammars in 4 categories :-

①	Type-3	Grammar	R.L
②	Type-2	"	C.F.L
③	Type-1	"	CSL
④	Type-0	"	R.E

A → αB/β right linear
A → Bα/β left linear

TYPE-3

A grammar in which all the productions are of type

$$A \rightarrow \alpha B / \beta$$

T3 RE
T2 CFL
T1 CSL
T0 RE

$\alpha, \beta \in T^*$ (terminal)
$A, B \in V$ (variable symbol)

$A \rightarrow \alpha$

WONDER CEMENT

$A \to B\alpha \mid \beta$
$A \to \alpha B \mid \beta$

Date :

More specifically it is called as right linear grammar

$A \to aB \mid b$
$B \to cD \mid e$ ✓

$A \to aBb$ ✗

OR

A grammar is said to be Type-3 if all its productions are of type

$A \to B\alpha \mid \beta$

& more specifically it is called as left linear grammar

⇒ Ya to left linear ho ya right linear in dono ka combination nahi hona chaiye

WONDER CEMENT

Date :

Eg

$A \rightarrow aA \mid bB \mid b$
$B \rightarrow bB \mid b$
} Right L.G

$A \rightarrow Aa \mid Bb \mid a$
$B \rightarrow Bb \mid a$
} LLG

$A \rightarrow aA \mid Ab \mid a$ → not Type 3

⇒ Type 3 grammar corresponds to regular languages. or it is a generator of R.L & corresponding machine is finite automata.

Type 2

If all the productions in a grammar are of type $A \rightarrow \beta$

$A \in V$ left, par variable

$\beta \in (V \cup T)^*$ right par variable, terminal ka any comb.

WONDER CEMENT

Date

eg $A \rightarrow aAb$

↓

left hand side par exactly one variable!

right me kuch bhi ho.

$S \rightarrow aSAb$

S: one variable

aSAb: any no.

→ they are also called "Context free Grammar"

means I can replace a variable without worrying about its neighbours. (at any intermediate step)

Ram goes 2 Agra
too Agra
to Agra
} Before me sab same nai

Context means surrounding

⇒ The lang that corresponds to context free grammar is context free language & the machine that corresponds to Type 2 grammar is PDA or Push Down Automata

⇒ Some properties that belongs to this class of lang.

$w \in L(G)$

↓ whether

given a string, belong to language generated by that grammar or not.

Parsing

null introduce hope of contraction

① Null production → & we need to remove them.

② A → B → unit production ⇓ it just produce extra steps → so algo ka time badta

left we variable / right we variable

(jo kam ek step me hota tha ; will take no. of steps)

② remove unit productions

③ remove ~~unreachable states~~ and non reachable symbols

We remove ε to speed up the process of production

Date

Generating symbols : symbols which directly / indirectly derives a string

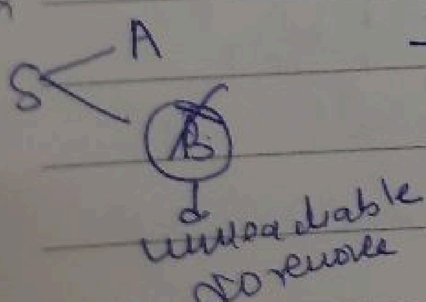
Also the symbols jo directly / indirectly kuch bhi strings nai generate karte unse kissi kaam k nai hai. & are called non generating symbols

(4)

S → aAS | bA ,

A → aA | a

~~B → bB | b~~



unreachable (~~cuz~~ cuz usse se final state par nahi aaya)

~~non gen~~

remove unreachable states

WONDER CEMENT

(5) left recursion

$A \longrightarrow A\alpha / \beta$

$w = \alpha^{100}\beta$

Parser will read $\longrightarrow$ $\alpha\alpha\alpha \ldots \beta$

problem problem of parser

In order to derive an algo we need to solve these problems first.

Context free Grammar

↓

Simplify

↓

Algorithm

WONDER CEMENT

Date :

① Removing null productions from the Grammar.

not all can be removed. If ϵ is a part of lang then we can't.

Consider a grammar

$$S \rightarrow aSb \mid AB$$
$$A \rightarrow \epsilon \mid aA$$
$$B \rightarrow \epsilon \mid bB$$

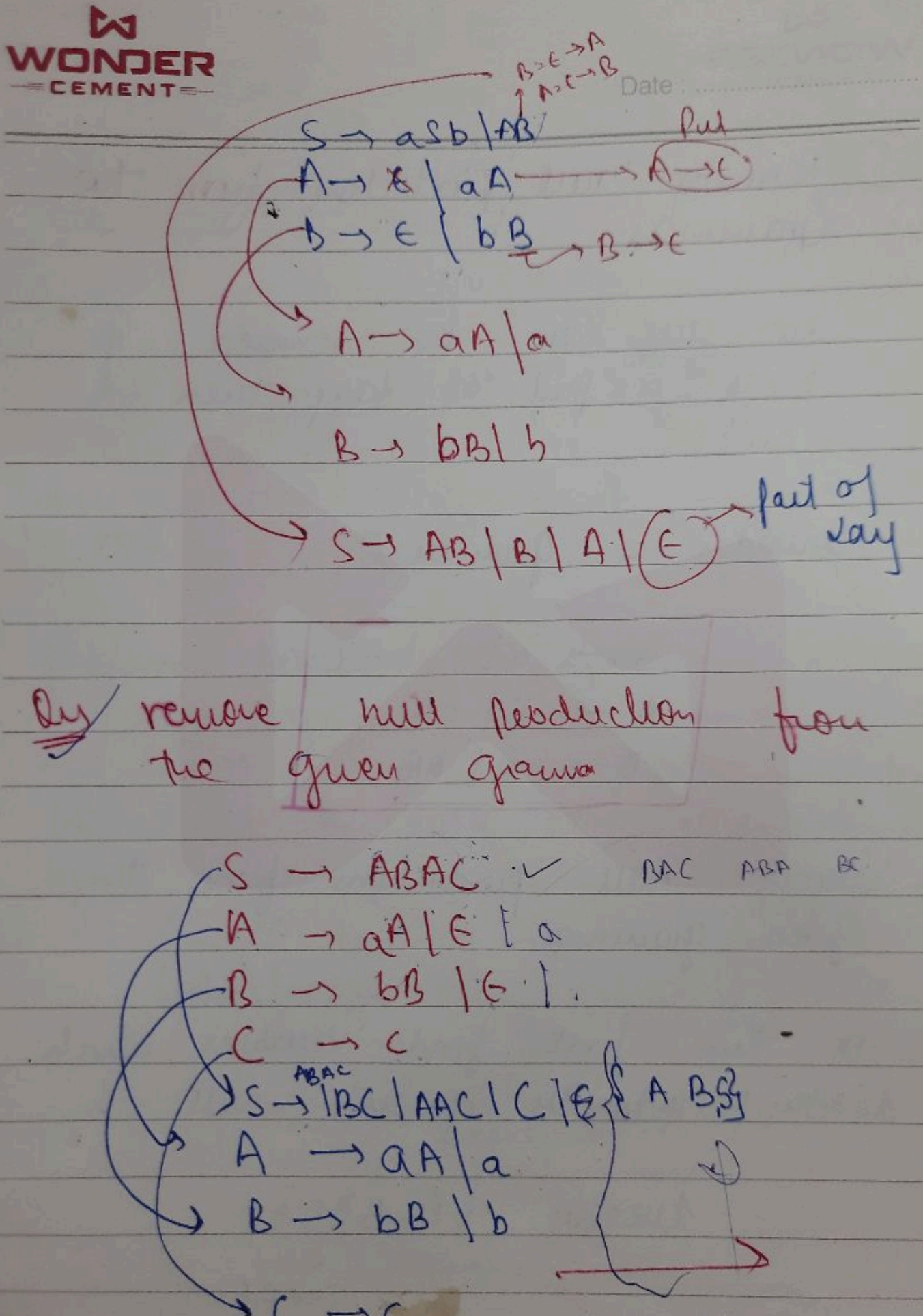
remove null productions from the given grammar.

For this first find variables which directly / indirectly generate null.

directly $\{A, B, S\}$

$S \Rightarrow AB$

$\epsilon \quad \epsilon$


S → aSb | AB
A → ∈ | aA
B → ∈ | bB
A → aA | a
B → bB | b
S → AB | B | A | ∈
part of say
Q) remove null production from the given grama
S → ABAC
A → aA | ∈ | a
B → bB | ∈ |
C → c
S → ABAC | BC | AAC | C | ∈
A → aA | a
B → bB | b
C → c

WONDER CEMENT

Date :

Direct by {A, B}

Pick one.

$S \rightarrow ABAC \mid BAC \mid ABC \mid BC \mid AAC \mid AC \mid C$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

$C \rightarrow c$

Q) Remove null production from the G

$S \rightarrow aSB \mid aB \mid a \mid Ab$

$B \rightarrow BB \mid aB \mid aA$

$A \rightarrow BA \mid a \mid \epsilon$ {A}

$S \rightarrow aSB \mid aB \mid a \mid b \mid Ab$

$B \rightarrow BB \mid aB \mid a \mid aA$

$A \rightarrow B \mid a \mid \epsilon \mid BA$

WONDER CEMENT

non generating

non reachable

Removing useless symbols → jisse directly ya indirectly terminal generate kar sakte hain

Q→ Consider a grammar

S → aS | A | C

A → a

B → aa ✗

C → aCb ✗ → it does not generate any terminal

A → se directly ek terminal string nikal rahi hai a.

B → se bhi ek string nikal rahi hai

{A, B} ∪ T

ab har production ko ginti me lao.

{A | B} ∪ T → that generate string

WONDER CEMENT

jo directly ya indirectly kisi kuch generate kar ke nahi deta use useless samajh k hata do.

C → non generating since it does not generate any thing (directly/ non directly)

remove C

do S → aS | A
A → a
B → aa

Now for non reachable (jo starting symbol se reachable na ho)

$S \overset{*}{\Rightarrow} \alpha X \beta$

→ A variable X is said to be reachable if $S \overset{*}{\Rightarrow} \alpha X \beta$

S se aS S → aS
S → A

S se B par reach nai kar pa rahe to B is unreachable

we can't reach B from S
remove B

S → aS | A
A → a

✓

WONDER CEMENT

Date :

Q1 Remove useless symbols from given CFG.

direct → S, E, F, B

S → aSB | aC | d

C → aC | DE

E → cc | DD

D → CA | aE

F → SC | Ba | a.

B → aA | a

A → BB | BC

A → an | a | e | c | b

B → b | c | a | e

C → C | a | e

D → ak

B A
F D
E
S

S → aSB

Directly production dete hai jese S, E, F, B,

(1) non reachable
(2) not found
(3)

Ye sab direct slang de rahe hai

A → indirect.

WONDER CEMENT

Date :

Unit Production :- $A \to B \to C \to D$

A ka set {BCD} → now BCD me jo jo non terminals hai unhe A me likh do.

B ka set {CD}

$S \to aSB \mid ac \mid d$

$E \to cc \mid DD$

$F \to Sc \mid Ba \mid a$ ✗ ye use nahi hoga

$B \to aA \mid a$

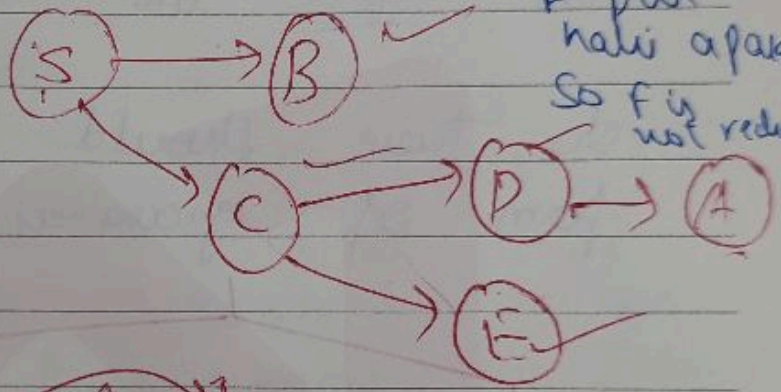
$A \to BB \mid BC$

$C \to$

$D \to$

S se F jane ka koi rasta nahi hai

S se koun F par nahi aparaha so F is not reachable



3. Unit Production

$A \to B \to b$

Ex $A \to aA \mid B \mid b \mid a \mid e$

$B \to b \mid C$

$C \to D \mid C$

$D \to a \mid e$

~~$A \to B \to C \to D \to a$~~

Bka

$A \to B \to C \to D$

$A = \{B, C, D\}$

$B \to$

$A \to B \to C \to D \to a \mid e$

$A \to aA \mid a \mid e \mid \cdot \mid c \mid e$

add non unit productions of all variables that are reachable to

WONDER CEMENT

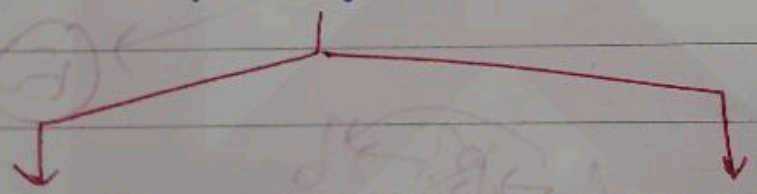
Date :

CNF

↳ Chomsky Normal form

We could have multiple grammars that represents same language. But the way, the grammar is written has effect on the way of parsing.

So there should be a proper form of grammar



CNF
Chomsky normal form

CNF
Greibach Normal form

If all the productions of grammar are of type

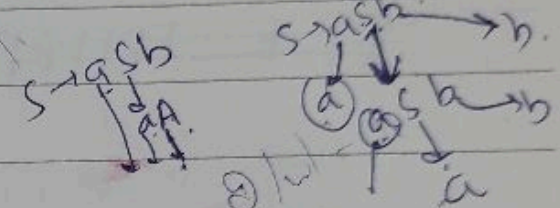
$A \rightarrow BC$
$A \rightarrow a$

WONDER CEMENT

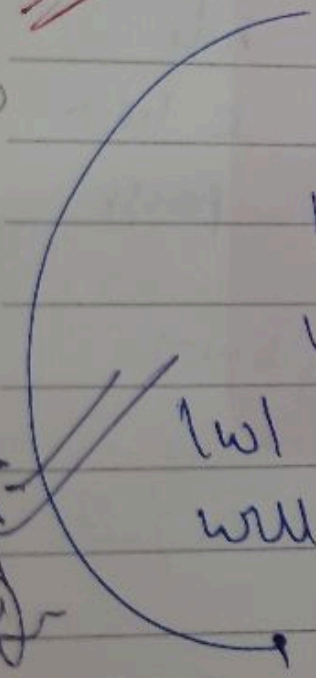
G → S → aSb | aA | bB | a. w = aaabb
A → aA | a
B → bB | b

derivation of w will be of length 2|w|-1 = 9

then the grammar is said to be in CNF.



done steps me | length badegi strings me.



S → BC
→ EFC
DCFC

|w| → (1) → b coz rule me 2 variable hai

length of string |w|-1 + |w| → to replace a variable.

|w| → length ki string will be derived in 2|w| = 2|w|-1 time.

S → BC
BDE } |w|-1

S → BC
C → DE

|w| { aDE, abE, abc

P → no of production
|P| → total no of prod.

S → aSb | bSa | bSb | aSa | a

~~|P| + |P||P|~~ S → aSb → out of 5 possibilities
↓
uske lie
bhi 5 possibility
hai

second step k lie
25 possi

$|P| + |P| \cdot |P| + |P|^3 + \dots |P|^{2w-1}$
↓
forms a GP.

$= O(|P|^{2w})$ → CNF time

$O(P^{2w})$

$O(n^3)$

$O(P^{2w})$

$O(P^{2w})$

WONDER CEMENT

CYK → membership

$O(n^3)$

Date :

$O(4^{2n})$

↓

4 productions

exponential complexity

or CYK $O(n^3)$

CNF (1) (Application)

It is used in CYK algorithm

It is very less efficient & based on dynamic (DAC) programming

$(P)^{2n}$

$w \in L(G)$

↳ $O(w^3)$ → string ki length uska cube kare par milegi

↓ CYK time

Input of CYK algo

↓

grammar in the form of CNF

⇒ CYK algorithm is used to find the membership of CFG

∴ Also its complexity is $O(n^3)$

WONDER CEMENT

Date :

* If grammar is in CNF then parse tree will be of Binary tree

CNF is good.

CYK is a universal algo, kissi par bhi laga sakte hai

→ O(|w|-1) → advance algo.
→ O(CP/2w).
→ O(w3) CYK
→ CNF form Binary tree

WONDER CEMENT

Date : .....

Ambiguity

↳ Confusion

10/2 * 4

→ 20

1 ←

} → Same expression, we get 2 outputs

Left se compute karne pe 20

right " " " " 1

a/b * c → string

If string has 2 O/Ps means it can be derived 2 ~~to~~ parse tree

⇒ If a grammar fails to provide unique structure of the sentance then grammar is ambiguous grammar

WONDER CEMENT

Date

* If there is a string w, that can be derived into two or more different ways that string is ambiguous string & grammar is ambiguous grammar.

⇒ There is no (algo) way we could determine whether a grammar is ambiguous or not.

⇒ We cannot remove the ambiguity in every grammar.

⇒ Inherently ambiguous

If a problem cannot be solved matlab kitni koshish karlo ~~to~~ esa hi rhega it cannot be solved → then it that problem is inherently ambiguous.

WONDER CEMENT

Date :

It can't be solved.
→ Only by luck.
→ Not a trial.

* Ambiguity is an undecidable problem.

WONDER CEMENT

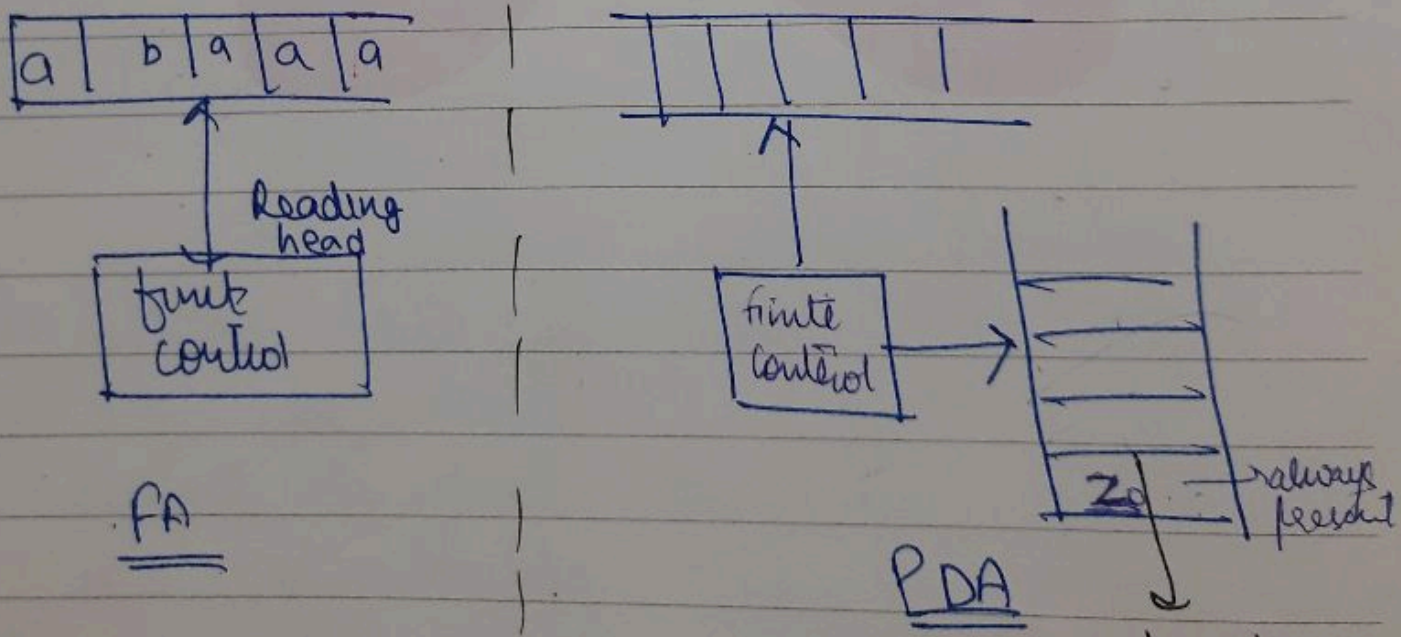
Date :

Pushdown Automata (PDA)

The device or machine that corresponds to context free languages is called as Push Down Automata.

Pushdown Automata	= finite automata	+ Memory element

↓ in form of 'STACK'



Kuch yaad rakhna hai to.

WONDER CEMENT

Date :

Stack :- Restricted Data Structure

* PDA is a 7 tuple collection

$(Q, \Sigma, \delta, q_0, Z_0, f, \Gamma)$

- Z_0 → marker to show bottom of stack or stack initial symbol it is a special symbol
- f → set of final states
- Γ → stack alphabet ie stack me kon kon se symbol kon it could be same as alphabet or diff. from alphabet.

$Q \times \Sigma \cup \{\epsilon\} \times \Gamma$

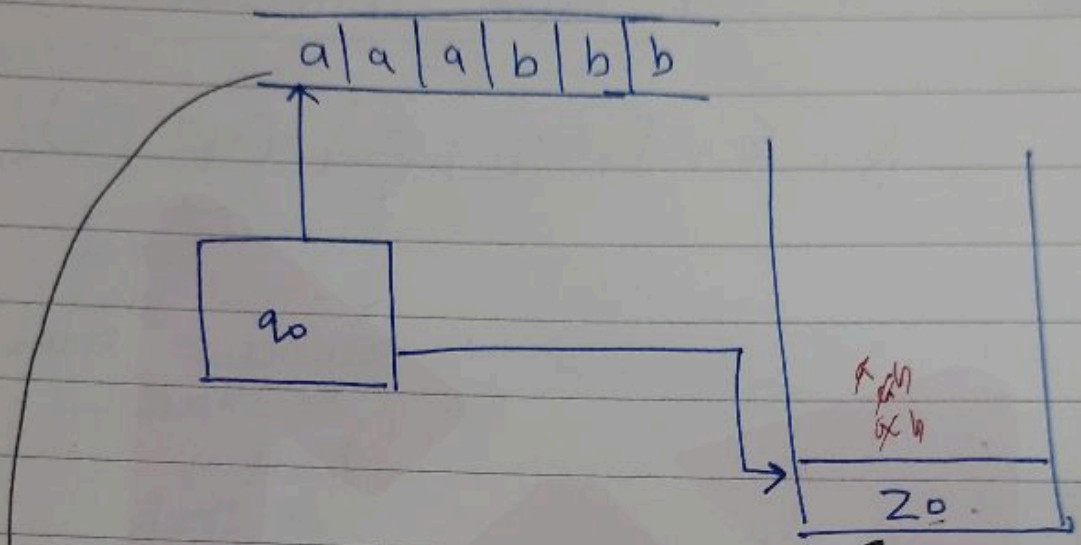
stack X reading direction X content of stack.

WONDER CEMENT

Date :

Let apple mango

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



Ye Z_0 mene remove kaiia hai & ye mere hath me hai

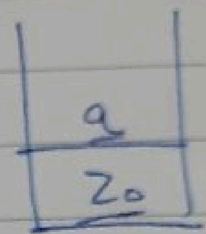
$Q \times \Sigma \times \Gamma$

$\delta(q_0, a, Z_0)$

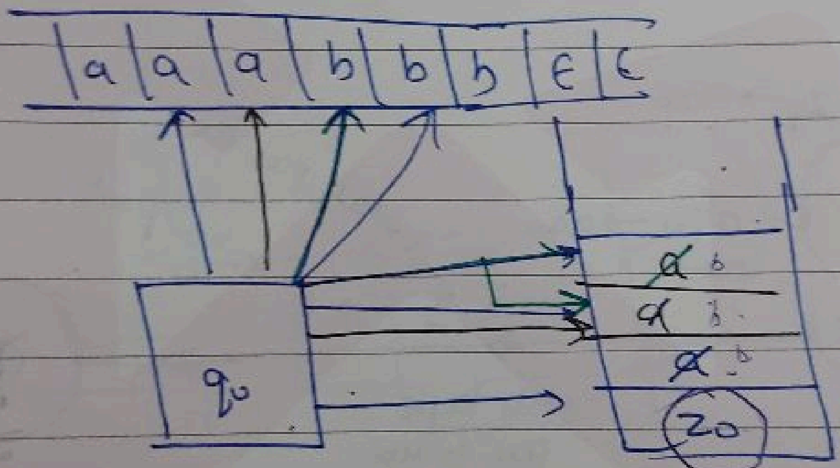
I am at state q_0 — reading a — element on stack

In order to push a, first push Z_0 then push a.

$= (q_0, aZ_0)$



Now



b atle hi maugo ajaya & purana a hatao pop & jo b uthaya use dalo mato

$Q \times \Sigma \times \Gamma$

$\Rightarrow \delta(q_0, a, a)$

q_0 par hu, a atla hai & stack par a hai

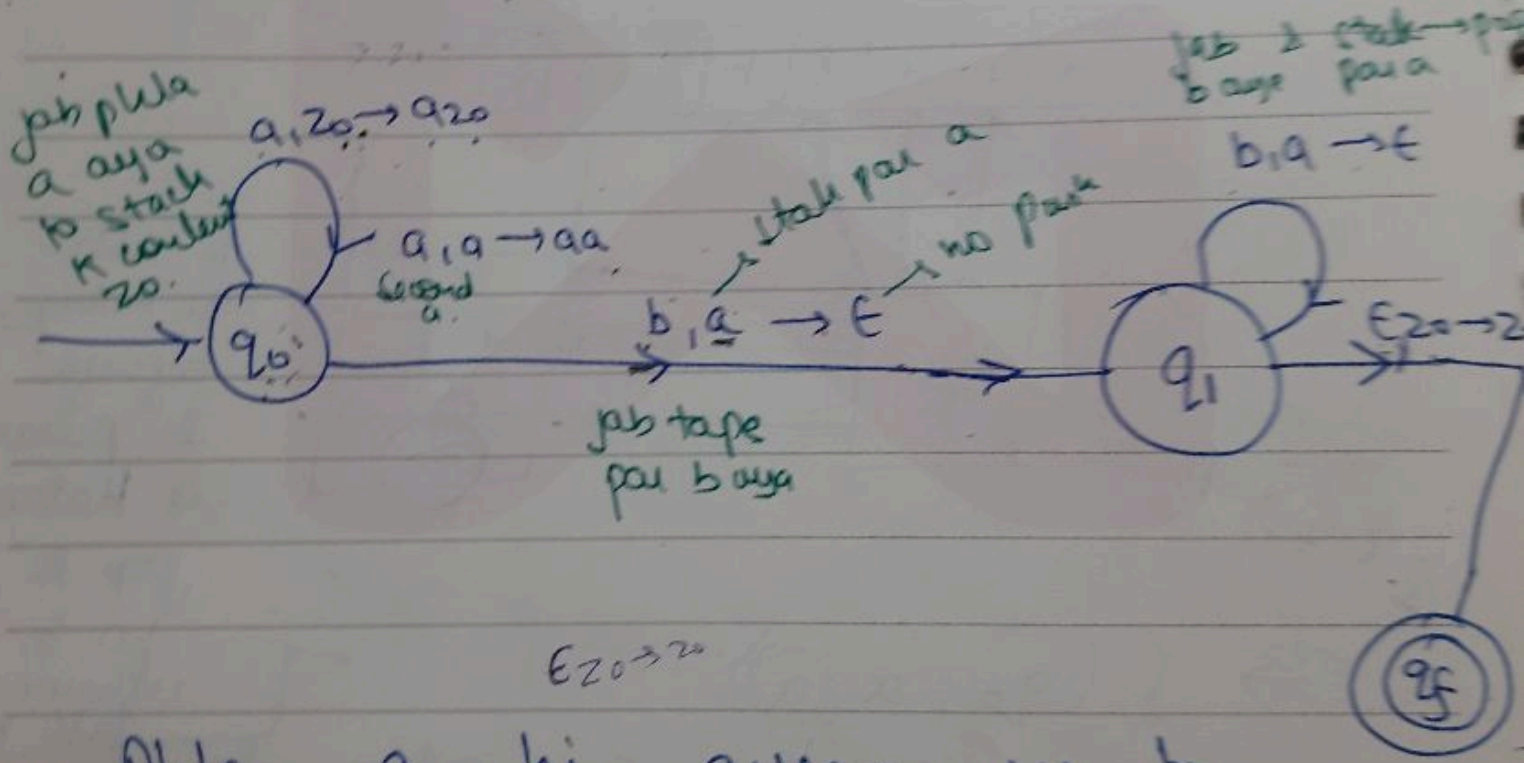
Length bad rahi hai Now then, it is push

$\delta(q_0, aa)$ Purana a ko utha lia

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

q_1 par hu — b aya — stack me a hai — null move (pop)

$\delta(q_1, \epsilon, Z_0) = (q_f, Z_0)$

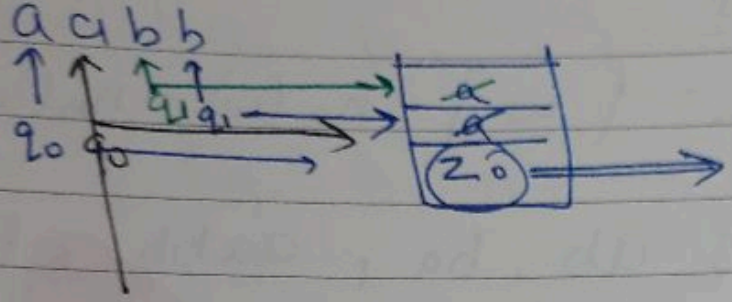


$\epsilon, Z_0 \to Z_0$

Pehle a hi aayega ye to confirm hai bcoz $a^n b^n, n \geq 1$

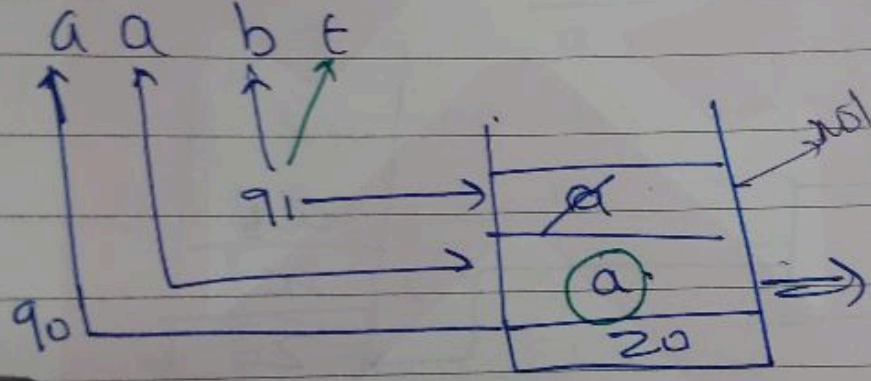
Date :

Lets take a stry aabb



= last me Z0 hi bach raha hai

final state acceptance



Machine hit a dead condition

Machine halts at a non final state.

rejectance

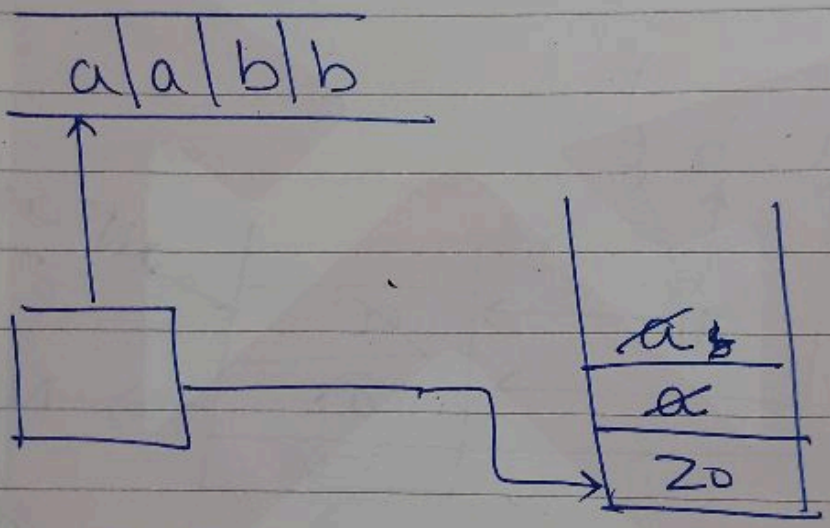
WONDER CEMENT

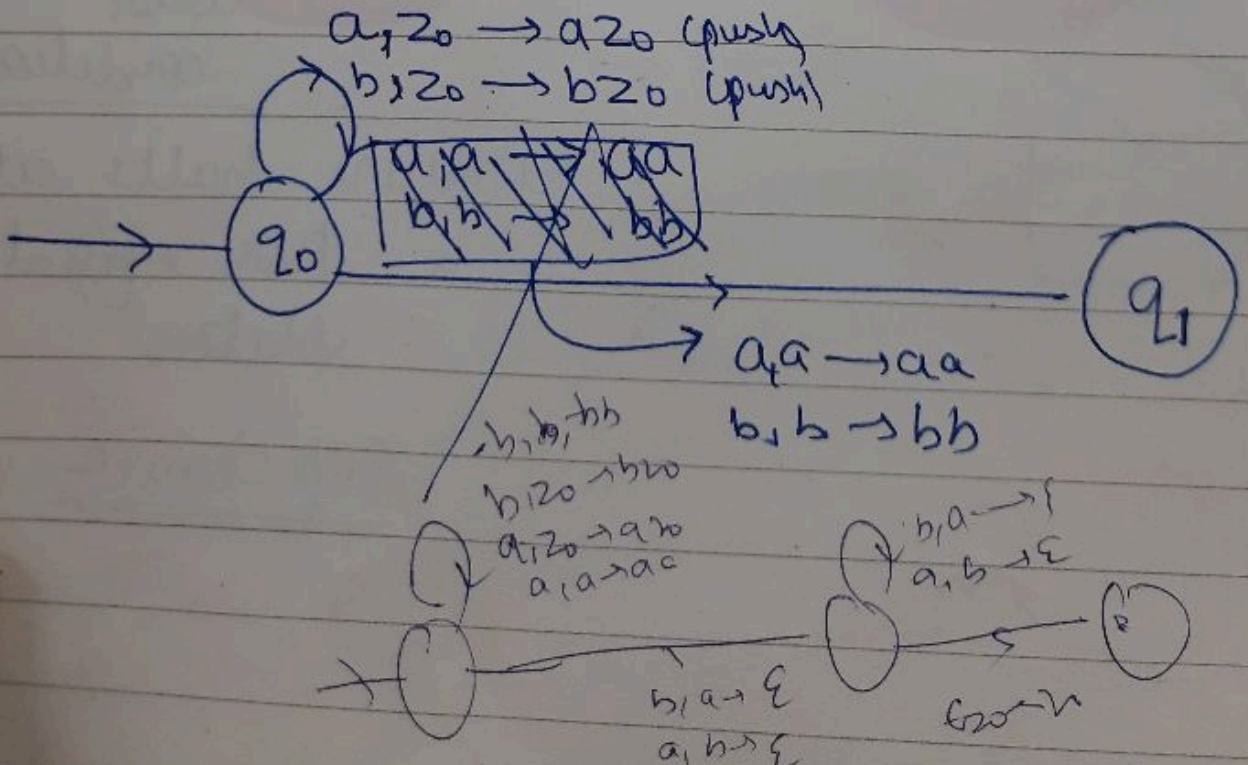
Date :

Ex: Design a PDA for the language

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

$\{ ab, ba, aabb, baab \}$





$\delta(q_0, a, z_0) \rightarrow \delta(q_0, \underline{a} z_0)$

$\delta(q_0, b, z_0) \rightarrow \delta(q_0, \underline{b} z_0)$

$\delta(q_0, b, a) \rightarrow \delta(q_0, \epsilon)$

$\delta(q_0, a, b) \rightarrow \delta(q_0, \epsilon)$

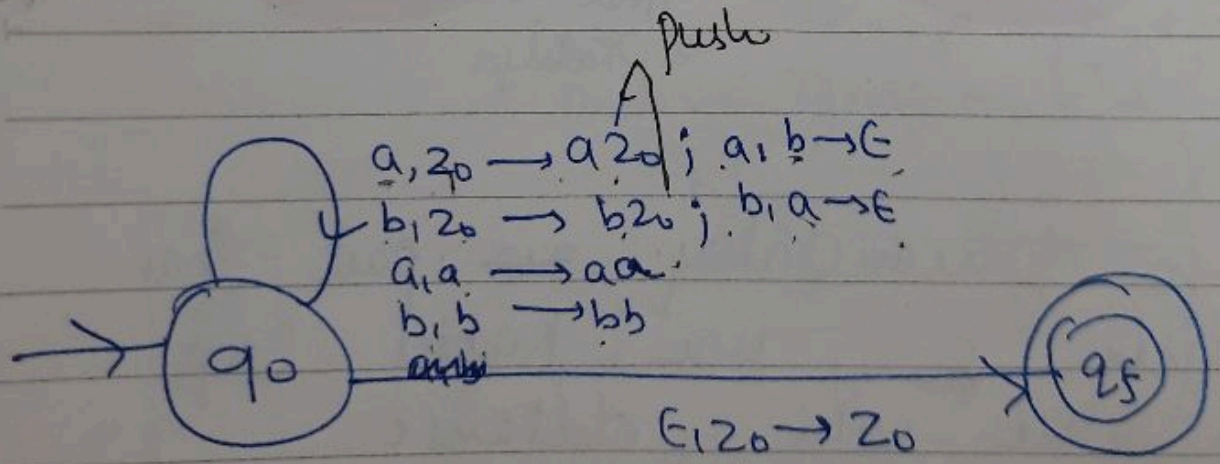
$\delta(q_0, a, a) \rightarrow \delta(q_0, aa)$

$\delta(q_0, a, b) \rightarrow (q_0, \epsilon)$

$\delta(q_0, b, a) \rightarrow (q_0 \epsilon)$

$\delta(q_0, b, b) \rightarrow (q_0, bb)$

$\delta(q_0, \epsilon, z_0) \rightarrow (q_f, z_0)$



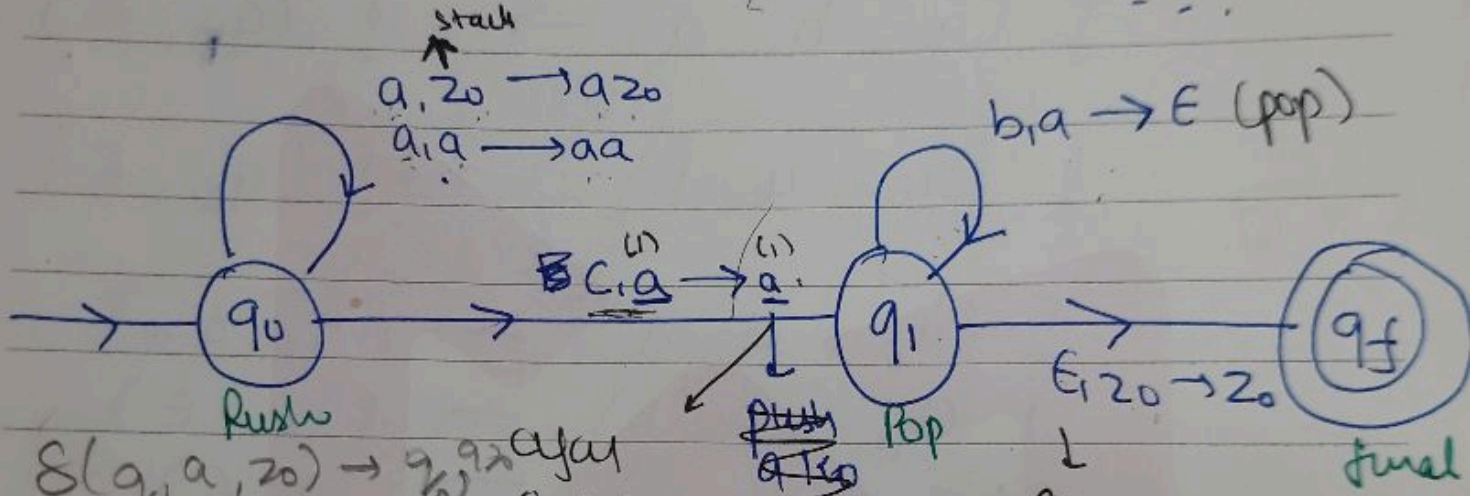
WONDER CEMENT

exactly 1 c

Q:- Design a PDA for the lang.

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

{acb, aacbb, aaacbbb - - -
. . .



$\delta(q_0, a, Z_0) \rightarrow q_0, aZ_0$ Agar c ayega to kuch nai karenge

$q_0, a, a \rightarrow q_0 aa$

$q_0, c, a \rightarrow q_1, a$ jo stack me tha wahi rahega

$q_1, b, a \rightarrow q_1, \epsilon$

$q_1, \epsilon, Z_0 \rightarrow$ Accept

Save a nikal jaenge

⇒ jo mandatory ~~me~~ part hai jese c, use kabhi loop me nahi, daleye

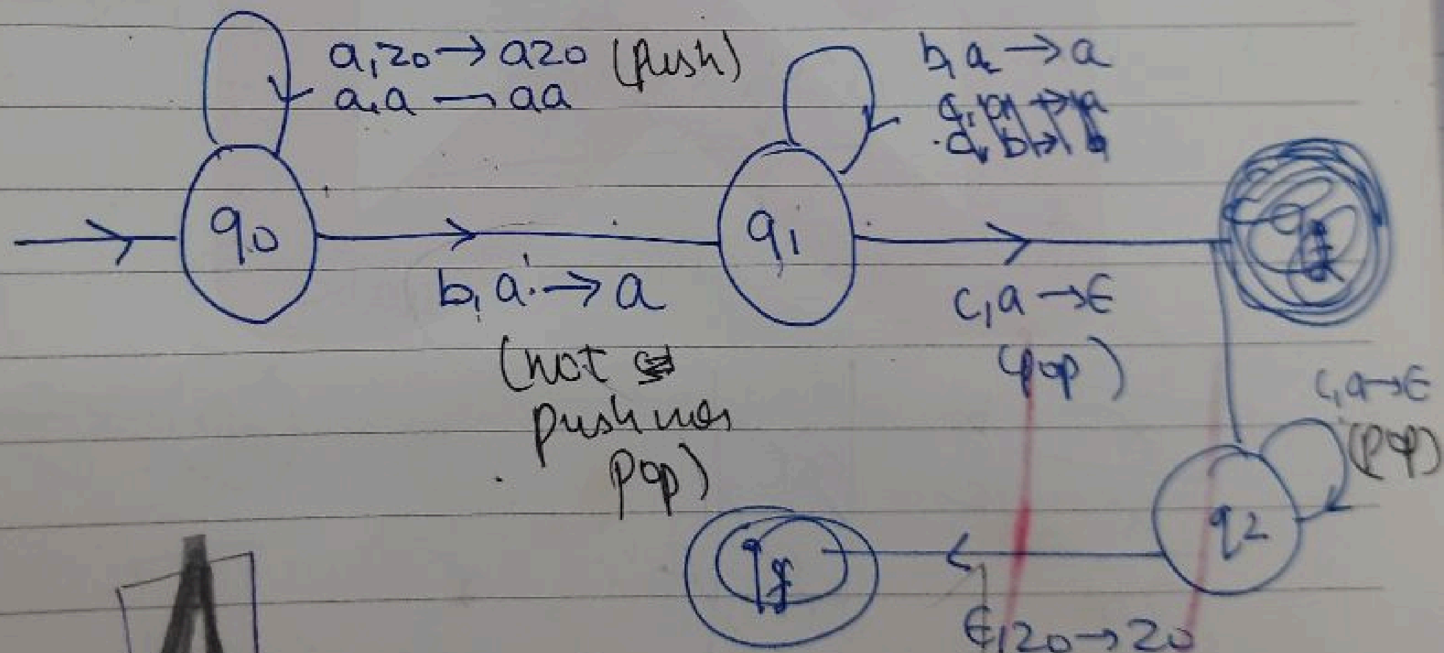
$c, a \longrightarrow a,$ → means jo hai usko read karo kuch naya mat karo

Stack of length 1

NOP no operation

Q. Design a PDA for the lang

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



$a^m b^n c^m$

$\delta(q_0, a, z_0) = (q_0, az_0)$

$\delta(q_0, a, a) = (q_0, aa)$

$(q_1, c, a) \rightarrow q_1, \epsilon$

$q_1, \epsilon, z_0 \rightarrow q_1, \wedge$

$(q_0, b, a) \rightarrow q_1, a$

$(q_1, b, a) \rightarrow q_1, a$

WONDER CEMENT

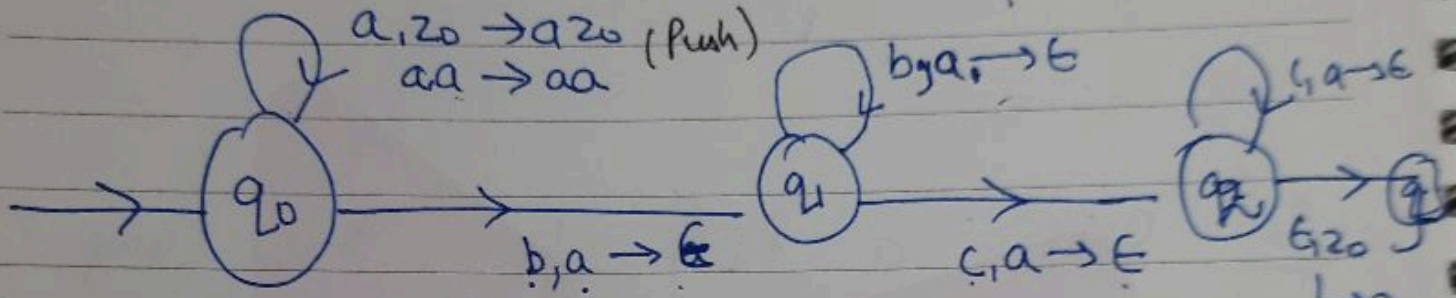
$a^m a^n b^m c^n$

Q Design a PDA for the lang.

$$L = \{ a^{m+n} b^m c^n \mid m, n \geq 1 \}$$

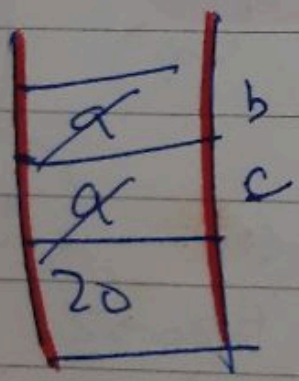
aabc

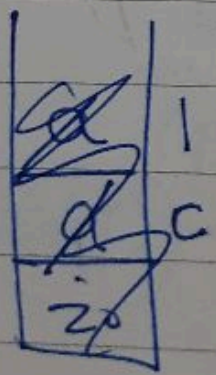
$a^m a^n b^m c^n$

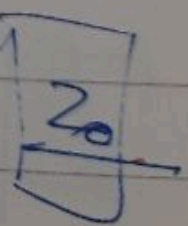


$a^m a^n b^m c^n$

aabc







WONDER CEMENT

Date :

Q Design a PDA for

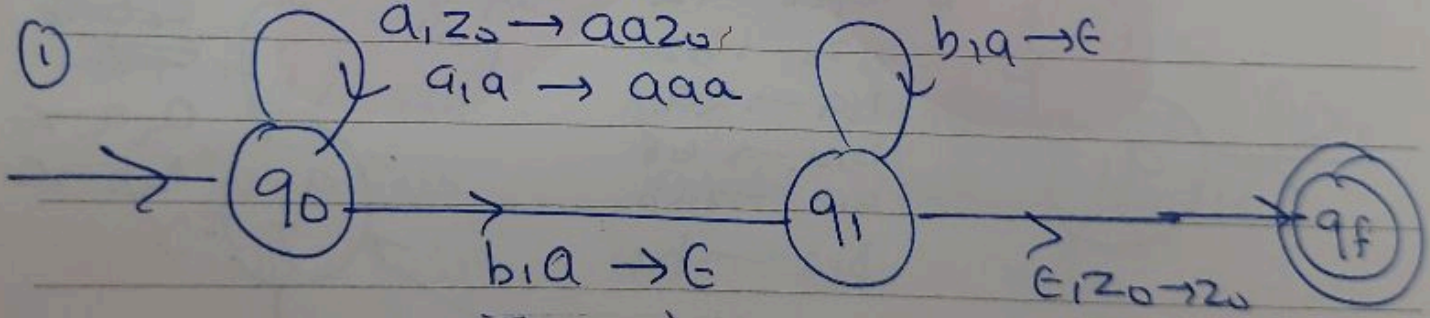
$L = \{ a^n b^{2n} \mid n \geq 1 \}$

abb

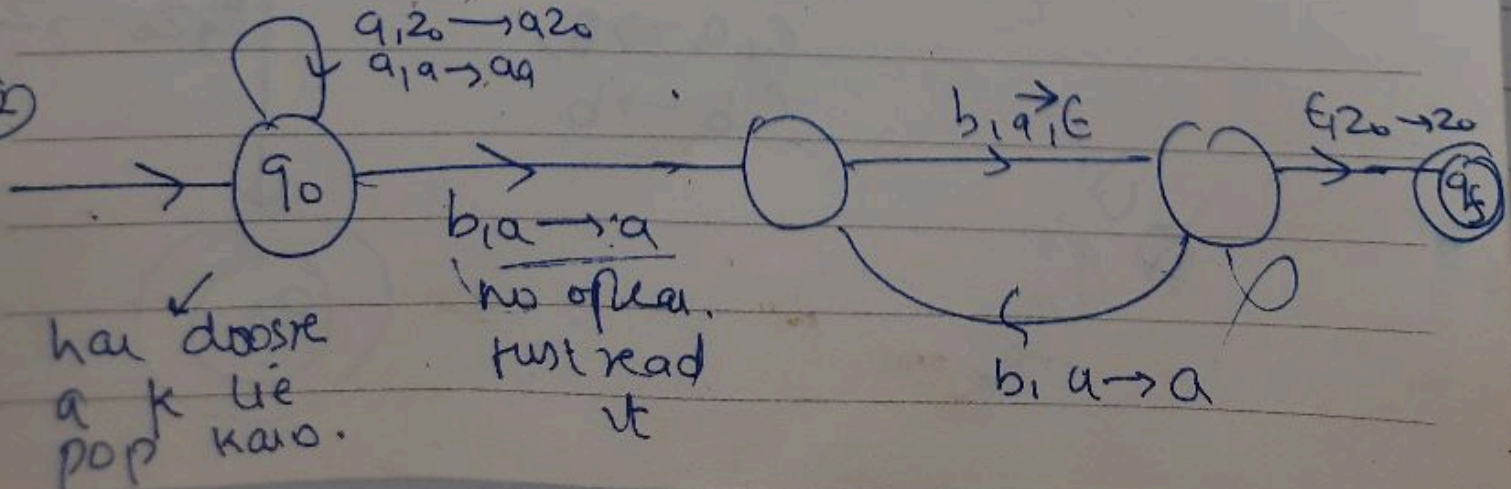
a

abb

(1)



(2)



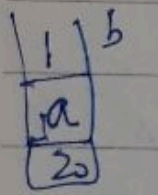
WONDER CEMENT

Date

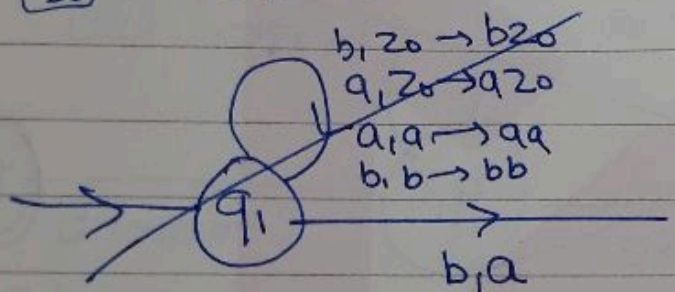
Design a PDA for a lang.

$L = \{ wcw^R \mid w \in (a,b)^+ \}$

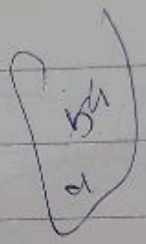
(wcw^R = story)

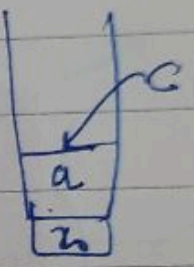


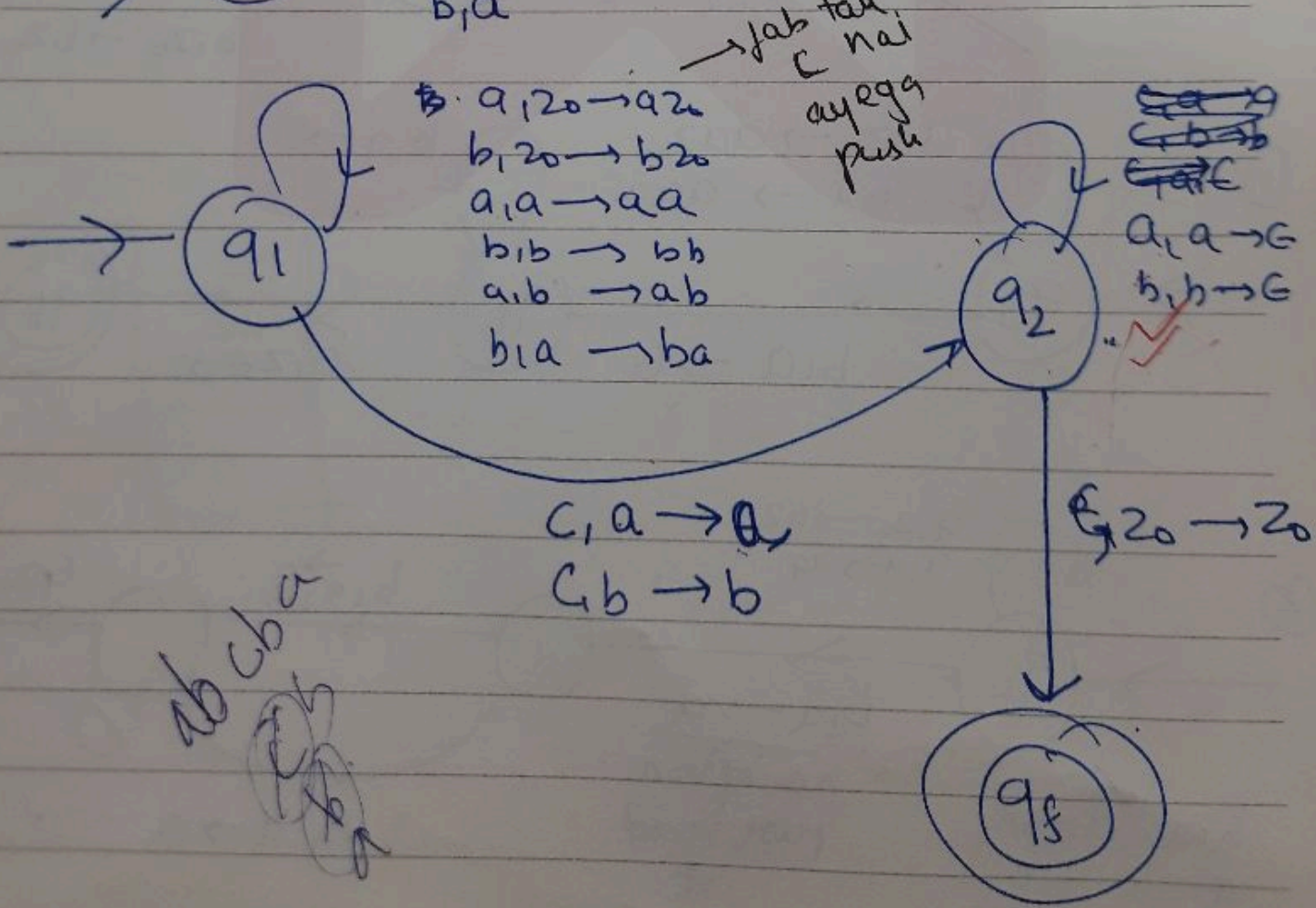
aca
abcba
bacab



abcba





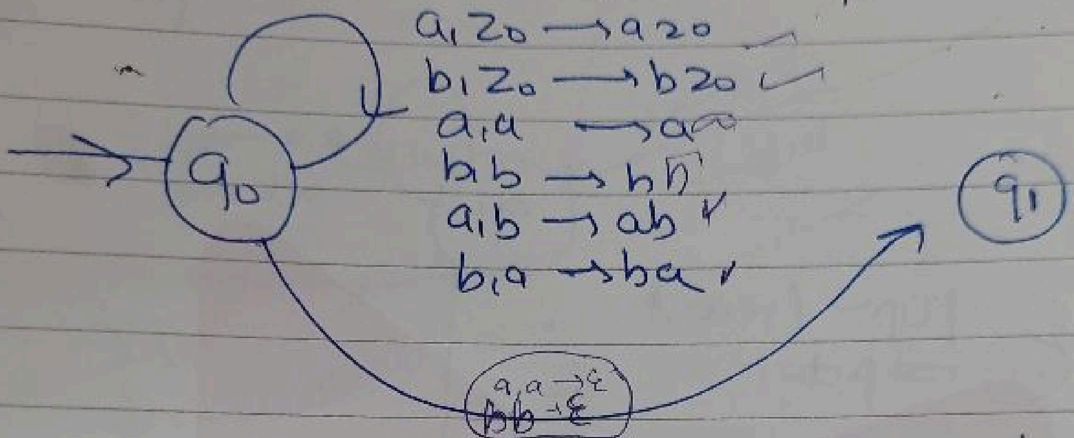


abcba

abba here pale hume dikhaye ki kis pop karne hai

Q→ Design PDA for

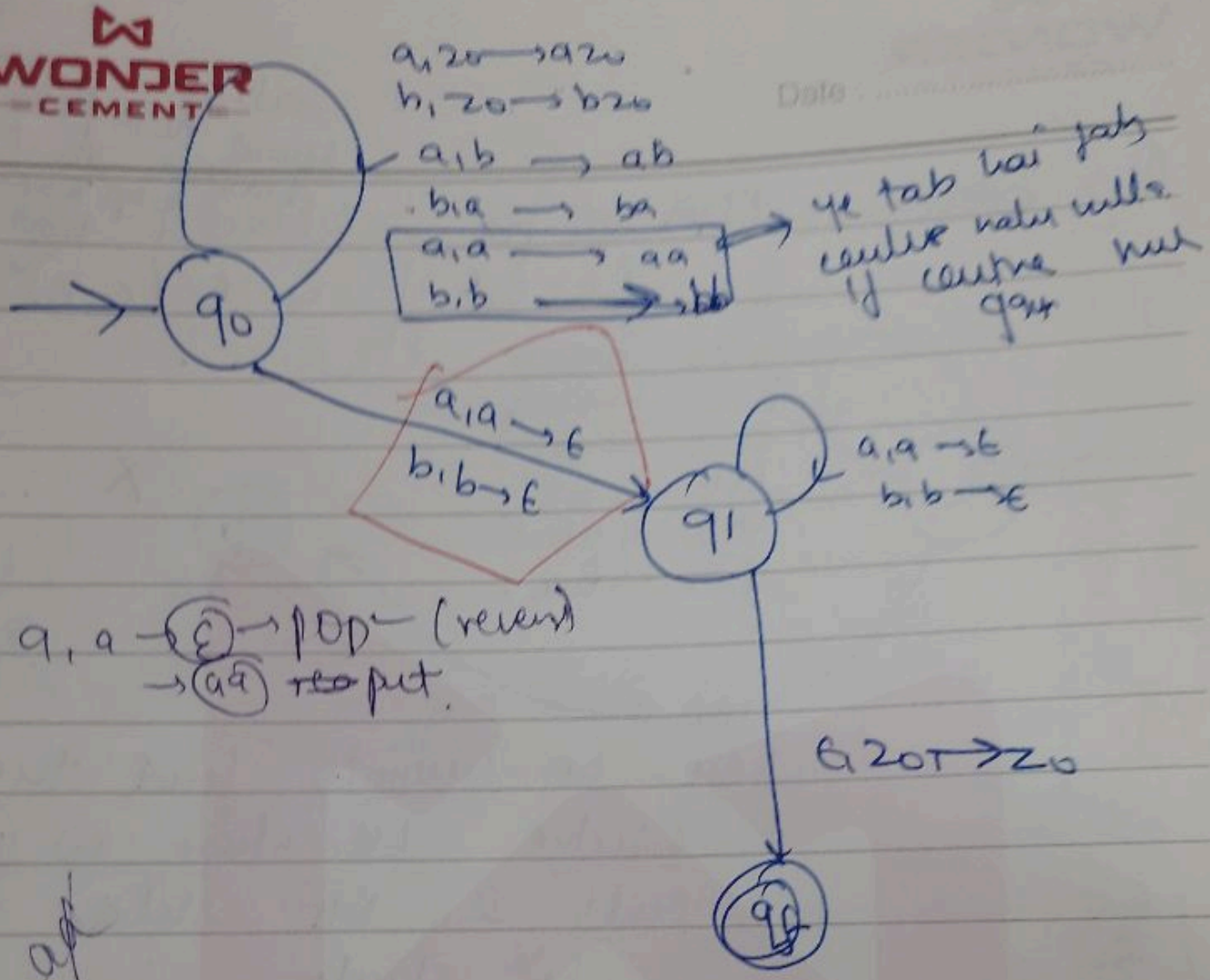
$L = \{ w w^R \mid w \in (a,b)^* \}$



abba

here we cannot find the centre till when w ends & till when W^R starts.

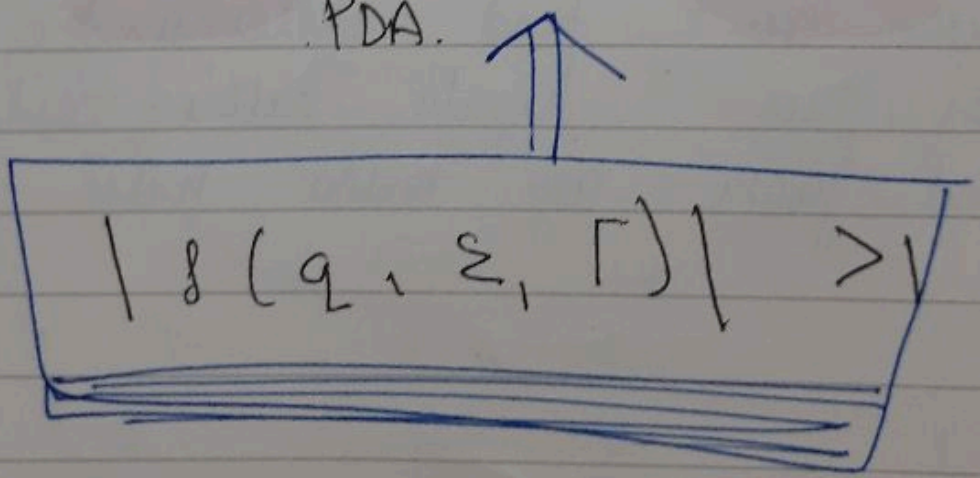
we can't find deterministic PDA for this kyunki kabhi centre mil jayega aur kabhi nahi milega.



q, a → (ε) → POP (recent)
→ (aa) ~~reput~~.

af

This is non deterministic PDA.



WONDER CEMENT

Date :

PDA of 9

* By default it is non deterministic

Let stmy is $(q_0, aaaa, Z_0)$ ← Stack

↓

ID → instantaneous description of a PDA

(q_0, aaa, aZ_0)

Centre | Centre not a

(q_1, aa, Z_0) (q_0, aa, aaZ_0)

(q, w, γ)

↓ current state

↓ remaining stmy to be read

dead configuration

halt

non final state rejection

Centre | Not centre

(q_1, a, aZ_0) $(q_0, aa, aaaZ_0)$

pop

Centre | Not centre

(q_1, ϵ, Z_0) (q_1, ϵ, aaZ_0) $(q_0, \epsilon, aaaaZ_0)$

q_1 dead non fin ↓ no more

There are some languages for which we cannot make deterministic PDA.

Deterministic PDA is less powerful.

Regular language ⟶ finite automata.

DCFL ⟶ DPDA

CFL ⟶ NPDA

→ deterministic context free lang. ⟶ deterministic PDA

→ context free language ⟶ non deterministic PDA.

~~NPA is more powerful than~~

* NPDA can't be converted into DPDA.

non deterministic push down automata can't be converted into deterministic push down automata.

NPDA & PDA have diff expressive powers

WONDER CEMENT

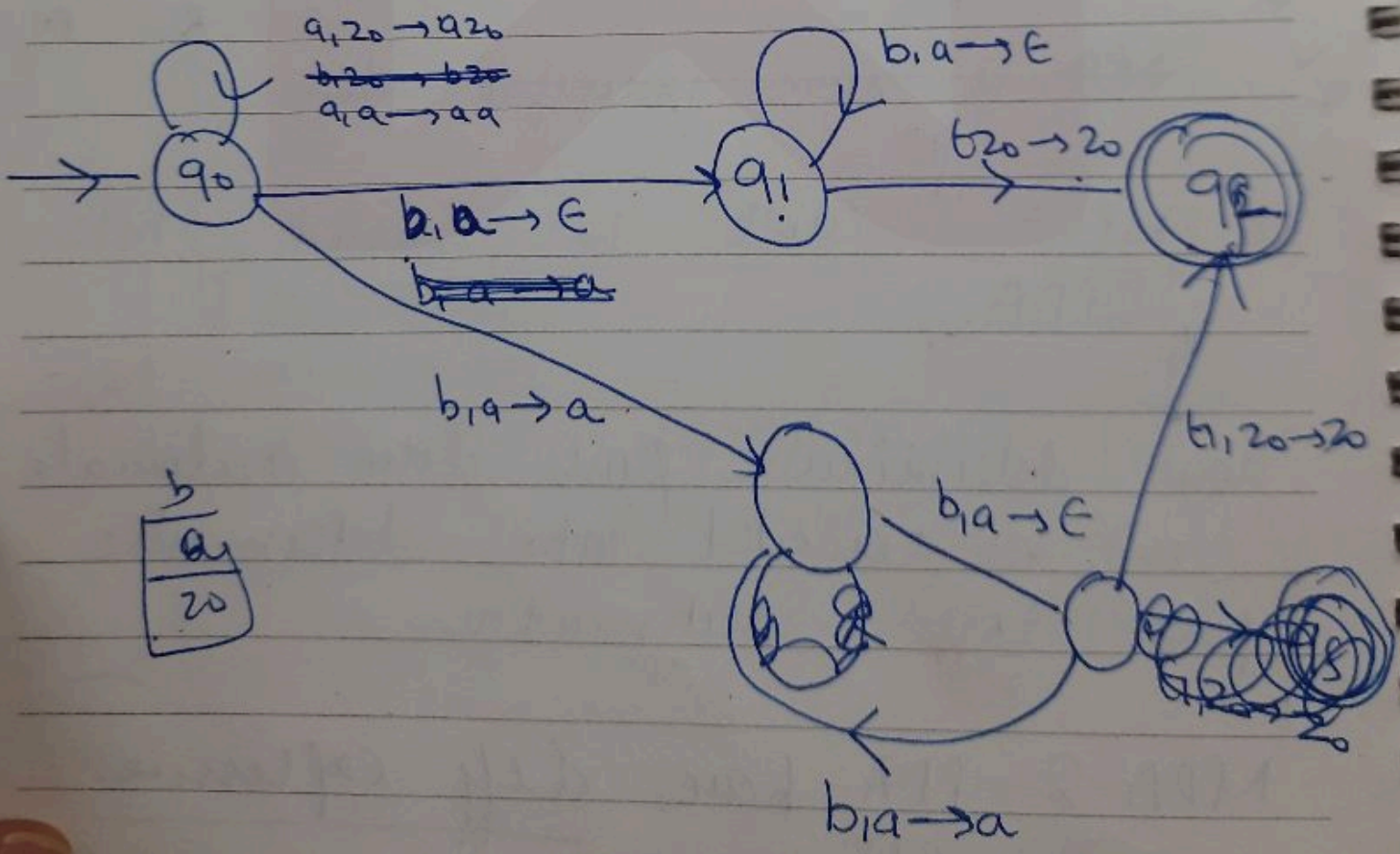
aabb

Ques Design a PDA for lang.

$$L = \{\{a^n b^n\} \cup \{a^n b^{2n}\} \mid n \geq 1\}$$

Deterministic is not possible as decision is to be taken b/w I & II.

So draw Non deterministic PDA



⇒ A lang (L) satisfies the pumping lemma for reg. lang it requires L is regular or non regular.

Date : 4/10

(L)

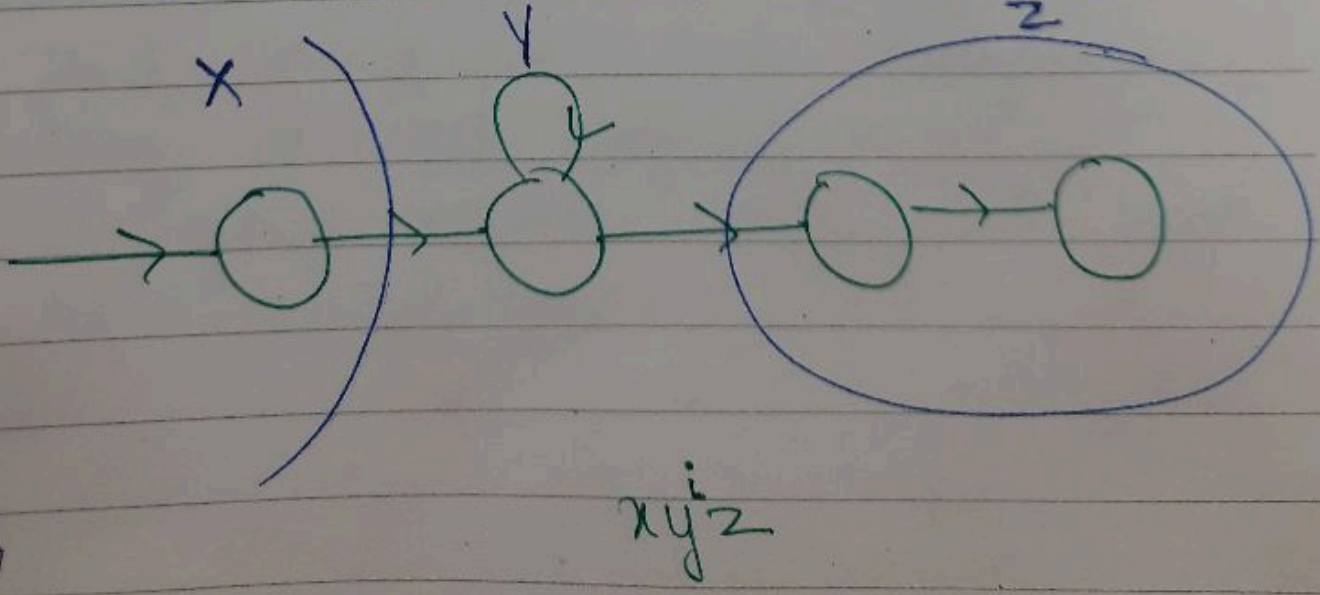
Pumping lemma (Not very imp!) only concept

Suppose DFA with 4 states with no cycle
So length of string zada se zada 3, only finite string with be accepted.

& agar ek bhi loop hai to lang. infinite ban jaegi

Based on Pigeon Hole Principal

also this loop is optional to ho kitni bhi baar aasakta hai



xy^iz

WONDER CEMENT

Date :

→ usme y loop hai

xy^iz → is type ki saari strings ban jaengi

It is a descriptive approach used to proved that language is 'not regular'.

Let L (lang) is regular → to DFA banega.
then strings will be of type
$w = xy^iz$.
Now we need to find esi value of i jiske lie string lang. me na ho, to hum prove kar denge it is not regular.

WONDER CEMENT

Date :

Other eg! -

$0^n 1^n$ → doping is done

$0^{n+k} (0^k)^2 1^n$

↓

$0^{n-k} 0^{2k} 1^n$

↓

$0^{n+k} 1^n$

NOTE

(1) Based on "Pigeon hole Principle."
(2) Used to prove that a given lang is not regular
(3) It gives negative result, or prove by contradiction.

WONDER CEMENT

Date

~~Turing Machine~~ as the

In
DPDA → (1) either a true I/P move or a null move for the same state an stack alphabet (Q, T)

(q_0, a, z_0) } only one can
(q_0, ϵ, z_0) } exist at a time

* If I/P move is defined we cannot define null move.

(2) $|\ |\ (Q, \Sigma, \Gamma)\ | \leq 1$

fully deterministic → DFA
Partially deterministic → PDFA
Non deterministic → multiple transitions on same I/P

WONDER CEMENT

DPDA

(deterministic pushdown automata) $Q \times \Sigma \cup \{\epsilon\} \times \Gamma \rightarrow Q \times \Gamma^*$

Γ^* → all of strings

stack ka element

NPDA

(non deterministic pushdown automata) $Q \times \Sigma \cup \{\epsilon\} \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$

$Q \times \Sigma \cup \{\epsilon\} \times \Gamma \rightarrow Q \times \Gamma^*$

$Q \times \Sigma^{\epsilon} \rightarrow Q \times \Gamma^*$

$Q \times \Sigma \cup \{\epsilon\} \times \{\epsilon\} \rightarrow Q \times \Gamma^*$

WONDER CEMENT

Date :

transition ↑ state

Mealy & Moore

Both of them are defined by (6) tuple.

$(Q, \Sigma, \delta, q_0, \Delta, \lambda)$

Δ → output alphabet

λ → output function (when to print the O/P)

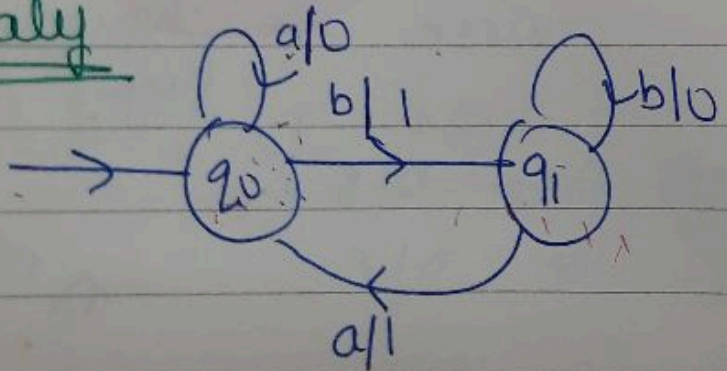
Moore associated to **state**

Mealy " " **transition**

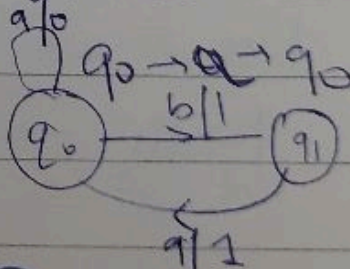
Moore :- $(n+1) \to s$

Mealy $n \to r$

Mealy

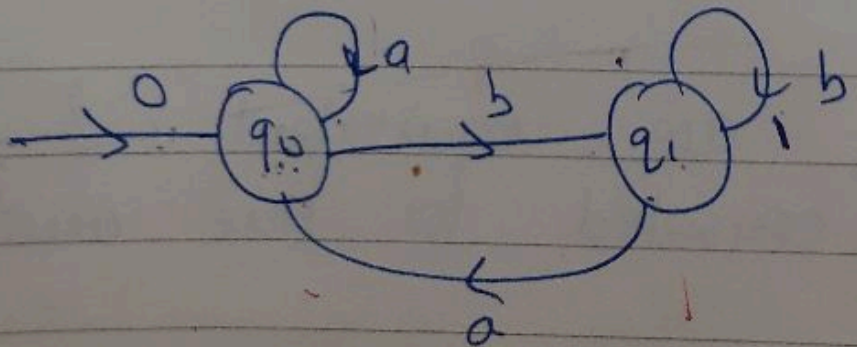


transition



Moore

(1)



clah

WONDER CEMENT

Date :

Moore me states ko jese $q_0 \to 0$
$q_1 \to 1$
assigned hotta hai

Mealy me transitions.

eg i/p → aab

In case Mealy Output = 0 0 1
a a b

In case Moore (n+1) 0 0 0 1
ε a a b

Moore me hmesha ε hoga.

* Moore → (n+1) size o/p.
Mealy → n size o/p

n^{n^2}

⇒ They are equivalent & can be converted to one another