PUSH DOWN AUTOMATA (PDA) -

A Pushdown Automata (PDA) is a way to implement a context free grammar (CFG) in a similar way we design finite automata for Regular grammar.

→ It is more powerful than FSM.
→ FSM has a very limited memory but PDA has more memory.
→ PDA = FSM + a stack.

★ **Basic Structure of PDA** - The basic structure of PDA is -
→ 'An input tape' - on which the input string is placed.
→ 'A read head,' - which reads the character in the input string one by one.
→ 'A stack', - also known as pushdown store, in the form of auxiliary memory.

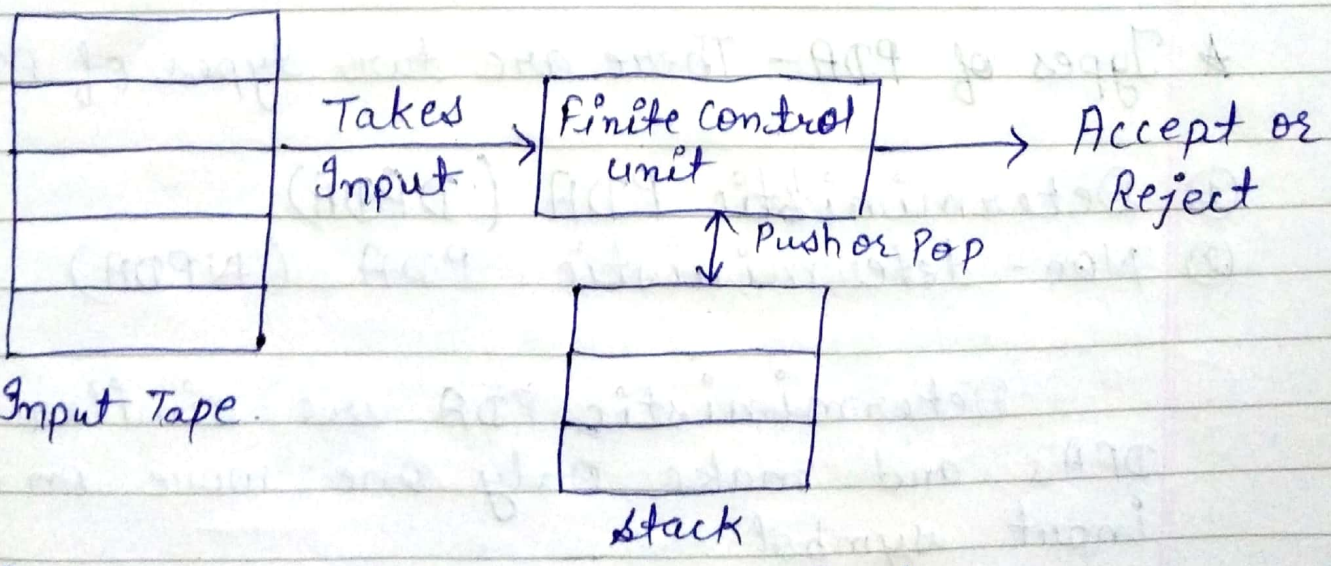


Fig - Basic structure of PDA

A PDA, $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a seven tuple machine.

where -

Q - finite non empty set of states.

Σ - finite non empty set of input alphabets.

Γ - set of symbols that can be pushed into the stack.

δ - transition function.

q_0 - starting state.

Z_0 - is initial pushdown store symbol. It is resident in the stack from its beginning.

F - set of final states. $\{F \subseteq Q\}$

The transition function is defined as -

$$\delta : Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma^*$$

where Γ^* is string formed from the stack alphabet.

★ Types of PDA - There are two types of PDA -

① Deterministic PDA (DPDA)
② Non-deterministic PDA (NPDA)

Deterministic PDA are similar to DFAs and make only one move on a input symbol.

Non-deterministic PDA are similar to NFA. DFA & DPDAs differ in a single aspect, DPDA allows the PDA to make null transition whereas a DFA doesnot allow null transition.

* Acceptance by PDA:- PDA has two ways to accept strings-

① By a final state
② By a null store

In case of acceptance by a final state, a string is accepted by the PDA if at the end of the string, the read head is in one of the final states.

In case of acceptance by a null store, a string is accepted by the PDA if at the end of string the push down store (stack) is empty.

Ex① Design a PDA to accept the language-
$L = \{0^n 1^n / n >= 0\}$

Sol.- The valid strings for the language are.

ε	if	$n=0$
01	if	$n=1$
0011	if	$n=2$
000111	if	$n=3$
⋮		⋮

So the language accept the string starts with 0's and followed by the same number of 1's in a ordered way.

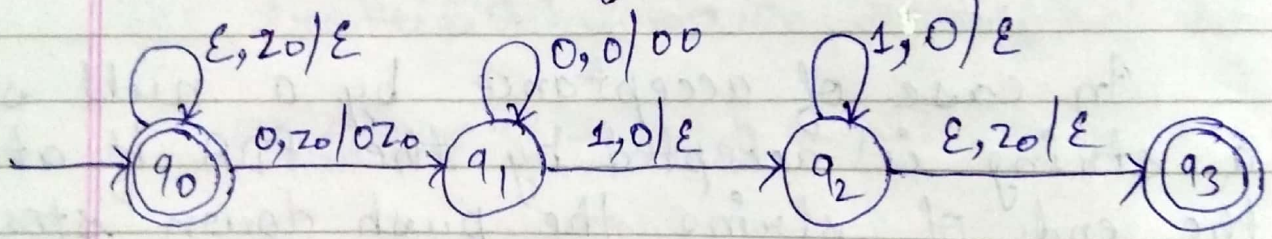
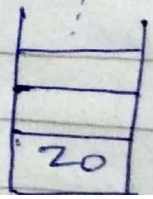


Fig- Transition diagram of PDA for language- $L = \{0^n 1^n / n >= 0\}$

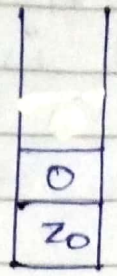
★ Behavior of the PDA on reading the string "0011" :-
The PDA is in the initial state q_0 before reading any symbol.

ⓐ Initially the stack contains only the initial stack symbol Z_0

PDA is in state q_0

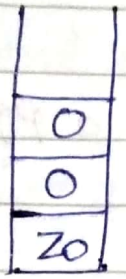


(b) After reading the first 0 in the string, stack:-



PDA is in state q_1

(c) After reading the second 0, stack is -



PDA is in state q_1

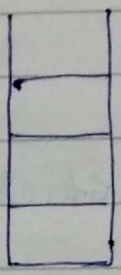
(d) After reading the third symbol (the first 1) in the string, the stack is -
(first 0 popeed off).

PDA is in state q_2

(e) After reading the fourth symbol (second 1), the stack is -
(second 0 is popped off).

PDA is in state q_2

(f) Top of the stack is now Z_0, therefore PDA can make a ε transition and move to state q_3, which is the accepting state.



PDA is in state q_3

* Instantaneous Description of a PDA on string 0011.

$$\delta(q_0, 0011, Z_0)$$
$$\downarrow$$
$$\delta(q_1, 011, 0Z_0)$$
$$\downarrow$$
$$\delta(q_1, 11, 00Z_0)$$
$$\downarrow$$
$$\delta(q_2, 1, 0Z_0)$$
$$\downarrow$$
$$\delta(q_2, \varepsilon, Z_0)$$
$$\downarrow$$
$$\delta(q_3, \varepsilon, \varepsilon)$$

ex② Design a PDA to accept the language
$L = \{a^n b a^n \mid n >= 1\}$

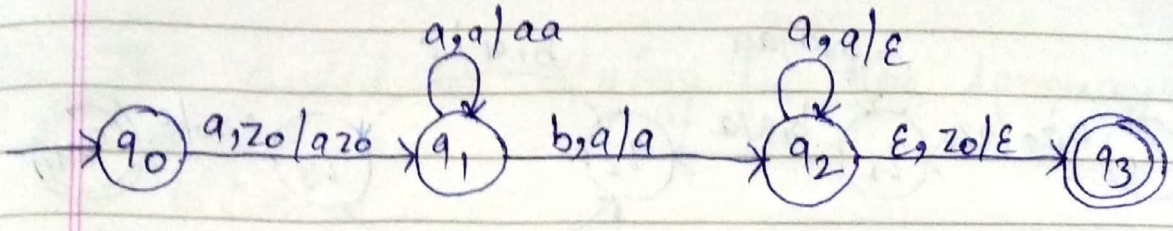


Fig- STD of PDA that accepts language
$L = \{a^n b a^n \mid n >= 1\}$

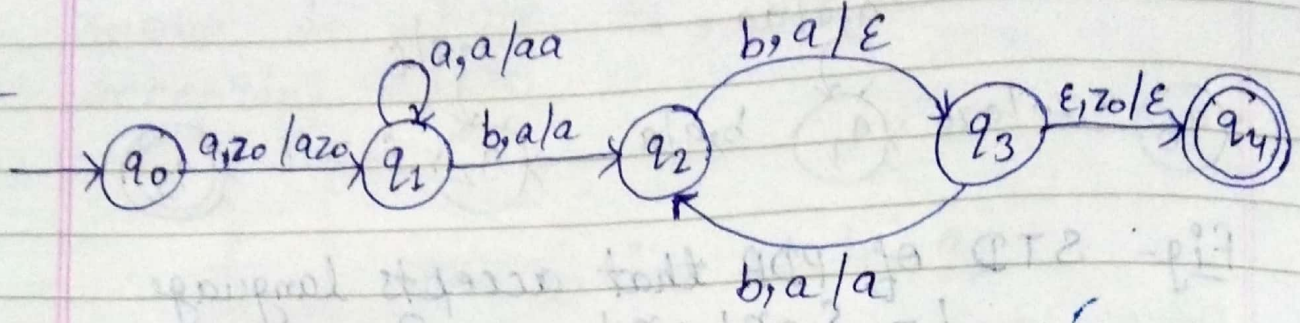
Content of the stack on reading string aaabaaa:-

Symbol read	State PDA	Content of stack
—	q_0	z_0
a	q_1	z_0 a
a	q_1	z_0 a a
a	q_1	z_0 a a a
b	q_2	z_0 a a a
a	q_2	z_0 a a
a	q_2	z_0 a
a	q_2	z_0
ε	q_3	

Ex ③ Design a PDA to accept the language

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

Sol-

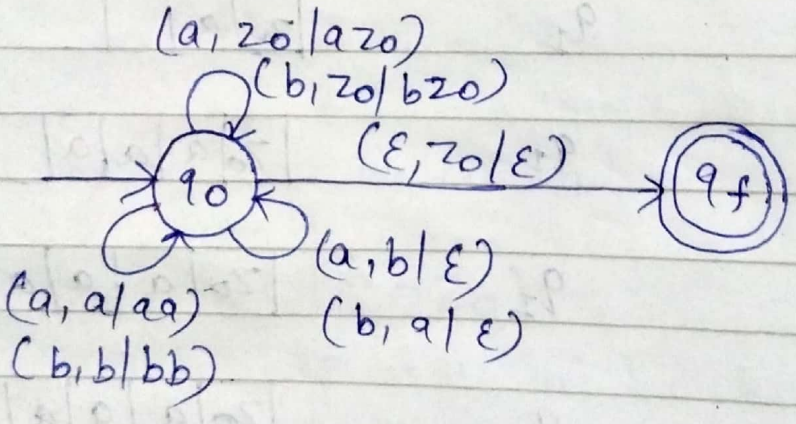


Ex ④ Design a PDA to accept the language -

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

number of a's and b's are equal.

Sol- $L = \{ ab, ba, aabb, bbaa, abab, baab \ldots \}$



Ex ⑤ Design a PDA for the language

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

Sol- The valid ex. string in the language are -

a c a

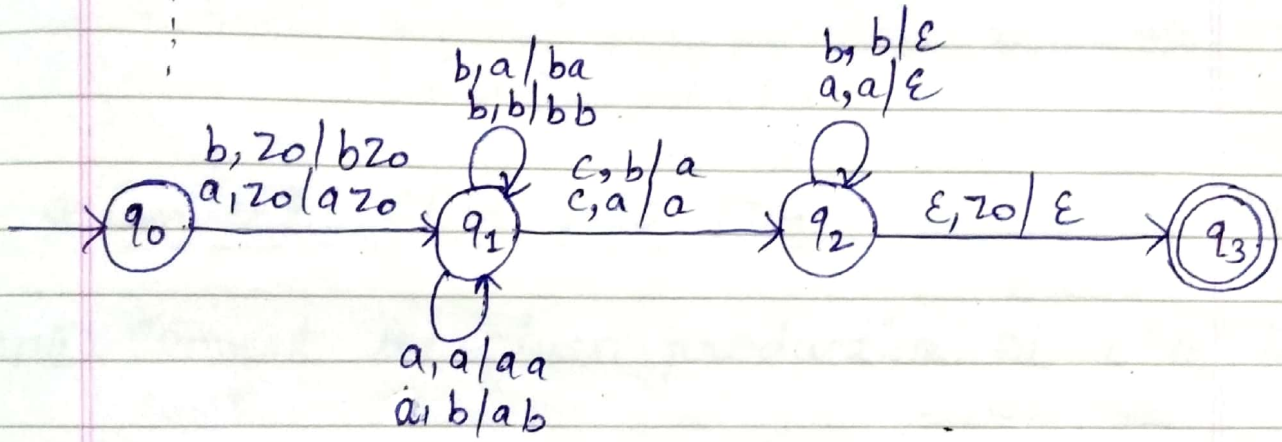
b c a

a b c b a

a a a c a a a

a b b c b b a

⋮



Ex ⑥ Design a PDA for the language -

$L = \{ a^i b^j c^k \mid i = j+k \text{ where } i, j, k >= 0 \}$

Sol- Valid string in language -

ε if $j=0, k=0$

ab if $j=1, k=0$

ac if $j=0, k=1$

aabc if $j=1, k=1$

aabb if $j=2, k=0$

aacc if $j=0, k=2$

⋮

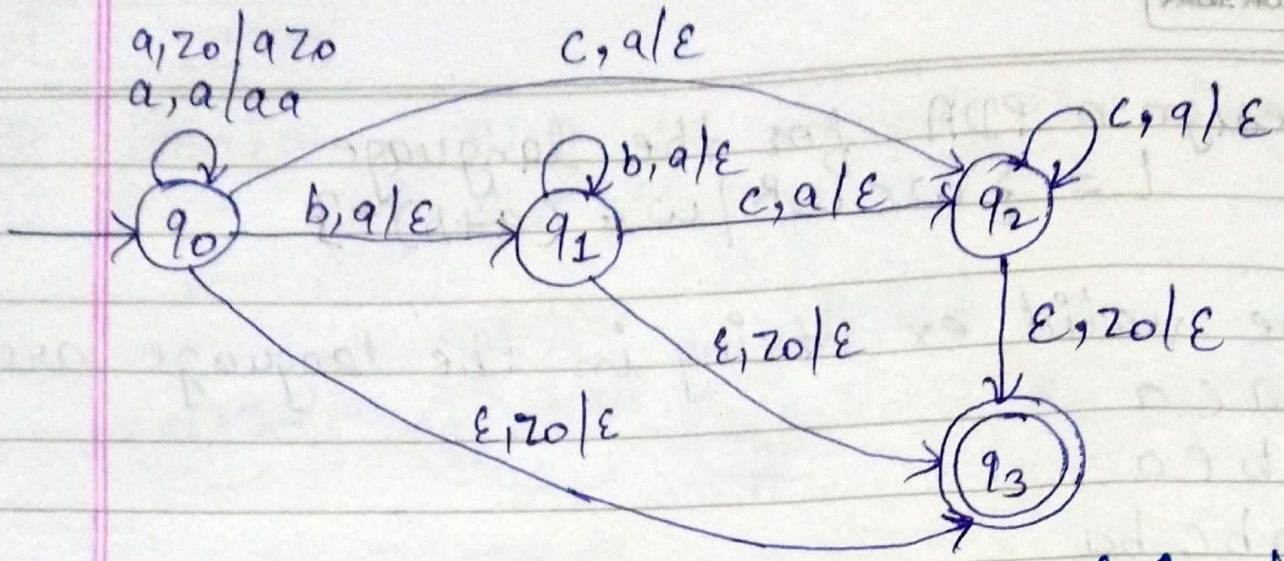


Fig- PDA for language - $L = \{a^i b^j c^k \mid i = j+k$ where $i, j, k >= 0\}$

✳ PDA and Context Free Grammar :-

If a grammar G_1 is context-free, we can build an equivalent PDA which accepts the language that is produced by the context-free-grammar G_1.

Also if P is a PDA, an equivalent context-free-grammar G_1 can be constructed where -

$$L(G_1) = L(P).$$

✳ Convert CFG into PDA -

Step ① Convert the given production of CFG into GNF.

Step ② The PDA will only have one state {q}

Step ③ The start symbol of CFG will be the initial symbol in PDA.

Step ④ For non-terminal symbol, add the following rule -

$$\delta(q, \varepsilon, A) = (q, q)$$

where the production rule is $A \to a$.

For each terminal symbol, add the following rule -

$$\delta(q, a, a) = (q, \varepsilon)$$

Ex① Convert a PDA from the following CFG-

$S \rightarrow OS1 | A$

$A \rightarrow 1AO | S | \varepsilon$

Sol.- First eliminate unit production-

$S \rightarrow OS1 | 1SO | \varepsilon$

Now convert CFG to GNF-

$S \rightarrow OSX | 1SY | \varepsilon$

$X \rightarrow 1$

$Y \rightarrow 0$

The PDA-

$R1: \delta(q, \varepsilon, S) = \{(q, OSX) | (q, 1SY) | (q, \varepsilon)\}$

$R2: \delta(q, \varepsilon, X) = \{(q, 1)\}$

$R3: \delta(q, \varepsilon, Y) = \{(q, 0)\}$

$R4: \delta(q, 0, 0) = \{(q, \varepsilon)\}$

$R5: \delta(q, 1, 1) = \{(q, \varepsilon)\}$

Ex② Construct PDA for given CFG-

$S \rightarrow OBB$

$B \rightarrow OS | 1S | 0$

Solution- The PDA can be given as-

$P = \{(q), (0, 1), (S, B, 0, 1), \delta, q, S\}$

The production rule δ can be:-

$R_1: \delta(q, \varepsilon, S) = \{(q, OBB)\}$
$R_2: \delta(q, \varepsilon, B) = \{(q, OS) | (q, 1S) | (q, 0)\}$
$R_3: \delta(q, 0, 0) = \{(q, \varepsilon)\}$
$R_4: \delta(q, 1, 1) = \{(q, \varepsilon)\}$

Consider the string 010000 -

$\delta(q, 010000, S) - \delta(q, 010000, OBB)$
$- \delta(q, 10000, BB)$ R1
$- \delta(q, 10000, 1SB)$ R3
$- \delta(q, 0000, SB)$ R2
$- \delta(q, 0000, OBBB)$ R1
$- \delta(q, 000, BBB)$ R3
$- \delta(q, 000, OBB)$ R2
$- \delta(q, 00, BB)$ R3
$- \delta(q, 00, OB)$ R2
$- \delta(q, 0, B)$ R3
$- \delta(q, 0, 0)$ R2
$- \delta(q, \varepsilon)$ R3
ACCEPT

So 010000 is accepted by the PDA.

☆ Nondeterministic PDA :-

A nondeterministic PDA (NPDA) is basically an NFA with a stack added to it. NPDA can make more than one move on an input symbol.

A NPDA is a 7 tuple structure -

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

Transition function for NPDA :-

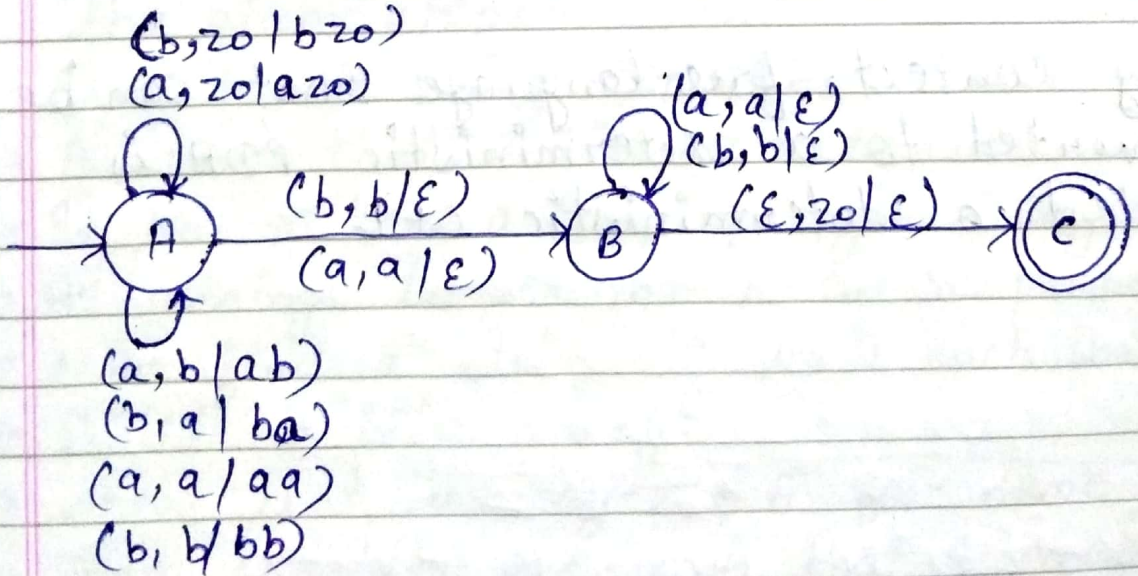
$$\delta : Q \times (\Sigma \cup \{\varepsilon\} \times \Gamma) \times \Gamma \to Q \times \Gamma^*$$

Ex ① Design a PDA for the language -

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

Sol.- if $w = ab$, $w^R = ba$ then

$ww^R = abba$



★ Deterministic PDA :-

In DPDA, there is at most one possible transition from any state based on current input.

Transition function for DPDA -

$$\delta : Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma^*$$

For every state P, input symbol 'a' and stack symbol t, there is at most one transition of the form -

$$\boxed{(P, a, t) \rightarrow (Q, u)}$$

for any state Q and stack symbol u.

Any context-free language that can be converted to a deterministic PDA is called a deterministic CFL.

DATE : / /
PAGE NO. :

Pumping Lemma for Context Free Language-

"Pumping Lemma (for CFL) is a technique for proving a language L is not a context-free language."

If A is a CFL, then A has a Pumping length 'P' such that any string 'S', where $|S| >= P$ may be divided into 5 parts - $S = uvxyz$ such that the following conditions must be true:-

① $uv^ixy^iz \in A$ for every $i >= 0$
② $|vy| > 0$
③ $|vxy| <= P$

To prove that a language is not context-free using pumping lemma (for CFL) follow the given steps-
(We prove using CONTRADICTION)

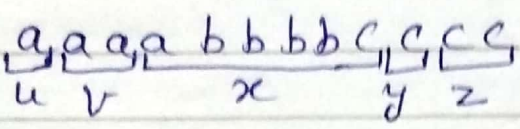
→ Assume that A is Context Free.
→ It has to have a Pumping length (say P)
→ All strings longer than P can be pumped $|S| >= P$
→ Now find a string 'S' in A such that $|S| >= P$
→ Divide S into uvxyz
→ show that $uv^ixy^iz \notin A$ for some i
→ Then consider the ways that S can be divided into uvxyz.
→ show that none of these can satisfy all the 3 pumping conditions at the same time.
→ S cannot be pumped == CONTRADICTION

Ex① Using Pumping lemma (for CFL) show that $L = \{a^n b^n c^n \mid n >= 0\}$ is not context free.

Sol. → Assume that L is context free
→ L must have a pumping length (say P)
→ Now we take a string s such that
$s = a^P b^P c^P$
→ We divide s into parts - uvxyz

Ex- $P = 4$ so $s = a^4 b^4 c^4$

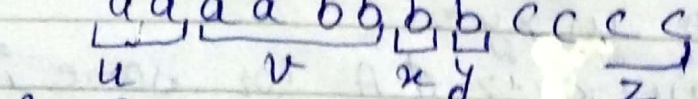
Case I- v and y each contain only one type of symbol.



$uv^i xy^i z \Rightarrow$ for $i = 2$; uv^2xy^2z

a aaaaa bbbbcccccc
$\Rightarrow a^6 b^4 c^5 \notin L$

Case II- Either v or y has more than one kind of symbols.



$uv^i xy^i z \Rightarrow$ for $i = 2$; uv^2xy^2z

aa aabb aabbb bb cccc $\notin L$

it

so, $L = \{a^n b^n c^n \mid n >= 0\}$ is not context-free.

Properties of Context Free Language:-

★ Closure Properties of CFL:-

① CFL are closed over union operation:-

Let L_1 and L_2 be two context-free languages. Then $L_1 \cup L_2$ is also context-free.

Let $L_1 = \{a^n b^n | n >= 0\}$ defined by $G_1 = (V_{n1}, \Sigma, P_1, S_1)$ and
$L_2 = \{ww^R | w \in (a+b)^*\}$ defined by $G_2 = (V_{n2}, \Sigma, P_2, S_2)$

The union of languages-

$$L = \{\{a^n b^n\} \cup \{ww^R\} | n >= 0, w \in (a+b)^*\}$$

defined by -

$$G = (V_n, \Sigma, P, S)$$

$$S \rightarrow S1 / S_2$$
$$S_1 \rightarrow aS_1b / \varepsilon$$
$$S_2 \rightarrow aS_2a / bS_2b / \varepsilon$$

Here L is context-free. so CFL are closed under union operation.

② CFL are closed under concatenation operation:
If L_1 and L_2 are context-free languages, then L_1L_2 is also context free.

Let $L_1 = \{a^n b^n / n >= 0\}$ defined by -

$$G_1 = (V_{n1}, \Sigma, P_1, S_1)$$

$$S_1 \rightarrow aS_1b / \varepsilon$$

and,

$L_2 = \{w w^R / w \in (a+b)^*\}$ defined by -

$$G_2 = (V_{n2}, \Sigma, P_2, S_2)$$

$$S_2 \rightarrow aS_2a / bS_2b / \varepsilon$$

Concatenation of the languages L_1 and L_2 -

$L = \{\{a^n b^n\} \cdot \{w w^R\} / n >= 0, w \in (a+b)^*\}$ defined by -

$$G = (V_n, \Sigma, P, S)$$

$$S \rightarrow S_1 \cdot S_2$$

$$S_1 \rightarrow aS_1b / \varepsilon$$

$$S_2 \rightarrow aS_2a / bS_2b / \varepsilon$$

Here language 2 is context-free.
So CFL are closed over the concatenation operation.

③ CFL are closed under Kleene closure operations
If L is a context free language, then L^* is also context free.

Let $L = \{a^n b^n / n >= 0\}$ defined by
$G_1 = (V_n, \Sigma, P, S)$.
$S \rightarrow aSb / \varepsilon$

Kleene closure of language L will be L^* or L_K with language.
$L_K = \{\{a^n b^n\}^* / n >= 0\}$
and with grammar.
$G_K = (V_n, \Sigma, P, S)$

$S_K \rightarrow S S_K / \varepsilon$
$S \rightarrow aSb / \varepsilon$

④ CFL are "not" closed over Intersection operation
If L_1 and L_2 are context-free languages, then $L_1 \cap L_2$ is not necessarily context free.

Let $L_1 = \{a^n b^n c^m / n, m >= 1\}$ defined by
$G_1 = (V_{n_1}, \Sigma, P_1, S_1)$
$S_1 \rightarrow XC / abc$
$X \rightarrow AXB / ab$
$C \rightarrow cC / c$
& $L_2 = \{a^m b^n c^n / n, m >= 1\}$ defined by-
$G_2 = (V_{n_2}, \Sigma, P_2, S_2)$
$S_2 \rightarrow AX / AbC$
$X \rightarrow BXC / bC$
$A \rightarrow aA / a$

Intersection of L_1 and L_2 is -

$$L_i = L_1 \cap L_2$$
$$L_i = \{a^m b^n c^m\} \cap \{a^m b^n c^n\} \mid n, m >= 1\}$$
$$L_i = \{a^n b^n c^n \mid n, m >= 1\}$$

The language L_i is not a CFL, so CFL is not closed over the intersaction operation.

⑤ CFL are "not" closed over complement operation-
If L_1 is a context free language, then $\bar{L}$ is not necessarily context-free.

This theorem is proved using proof by contradiction technique. Let us assume that CFLs are closed over the complement operation. Let there be two CFLs- L_1 and L_2 with their complement $\bar{L}_1$ & $\bar{L}_2$.

Since, we assume CFL are closed over complement therefore $\bar{L}_1$ & $\bar{L}_2$ are both CFLs. Their union $(\bar{L}_1 \cup \bar{L}_2)$ is also a CFL (Already proved). So, complement of $L_1 \cup L_2 = \overline{(\bar{L}_1 \cup \bar{L}_2)}$ is also a CFL.

$$\overline{(\bar{L}_1 \cup \bar{L}_2)} = L_1 \cap L_2.$$

We already know that intersection of CFLs is not necessarily a CFL.

So, our assumption is wrong & led to a contradiction. Therefore, CFLs are not closed over complement operation.

Note- $\overline{(\bar{L}_1 \cup \bar{L}_2)} = L_1 \cap L_2$ {De' morgan law}

⑥ CFL are "not" closed over difference operation
If L_1 and L_2 are context free then $L_1 - L_2$ is not necessarily context free.

Difference of two sets is denoted by $L_1 - L_2$. Let L_1 be Σ^* (standard CFL).
Then
$L_1 - L_2$ can be written as -

$$\Sigma^* - L_2 = \bar{L}_2$$

So $L_1 - L_2$ is not a CFL.

★ Decision Properties of CFL:-

① Test for Membership : Decidable
② Test for Emptiness : Decidable
③ Test for finiteness : Decidable.

① Algorithm to decide whether CFL is Empty or not:-

For a given CFG, there exists an algorithm to decide whether its language is empty $L(G) = \phi$ or not.

"If we can not derive any string of terminals from the given grammar then its language is called as an Empty language ; $L(G) = \phi$"

★ Algorithm:-

→ Remove all the useless symbols from the grammar.
→ A useless symbol is one that does not derive any string of terminals.
→ If the start symbol is found to be useless then the language is empty otherwise not.

Remember :- The language generated from a CFG is non-empty iff the start symbol is generating.

Example- Consider the following grammar-

$S \rightarrow XY$
$X \rightarrow AX$
$X \rightarrow AA$
$A \rightarrow a$
$Y \rightarrow BY$
$Y \rightarrow BB$
$B \rightarrow b$

check $L(G)$ is empty or not.

Solution-

$S \rightarrow aabb$
$X \rightarrow aX$
$X \rightarrow aa$
$A \rightarrow a$
$Y \rightarrow bY$
$Y \rightarrow bb$
$B \rightarrow b$

So,
→ The start symbol generates at least one string (many more are possible).
→ So start symbol is useful.
→ Thus, $L(G)$ is not-empty.

$$\boxed{L(G) \neq \phi}$$

② Algorithm to decide whether CFL is finite:-

For a given CFG, there exists an algorithm to decide whether its language is finite or not.

★ Algorithm:-

Step ① Reduce the given grammar completely by-
→ Eliminating ε production
→ Eliminating unit production
→ Eliminating useless production.

Step ② → Draw a directed graph whose nodes are variable of the given grammar.
→ There exists an edge from node A to node B if there exists a production of the form
$A \rightarrow \alpha B \beta$

Now following 2 cases are possible -

★ Case 1-
→ Directed graph contains a cycle
→ In this case, language of the grammar is infinite.

★ Case 2-
→ Directed graph does not contain any cycle.
→ In this case, language of given grammar is finite.

DATE : / /
PAGE NO. :

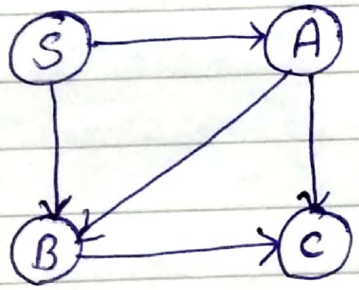
Ex① Check whether language of following grammar is finite or not -

$S \rightarrow AB \mid a$
$A \rightarrow BC \mid b$
$B \rightarrow CC \mid c$

Solution- Step ① - Given grammar is already completely reduced.

Step ② Draw a directed graph whose nodes - S, A, B, C

The graph is -



Here -
→ The directed graph doesnot contain any cycle
→ So L(G) is "finite"

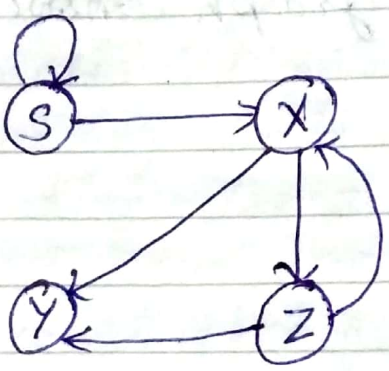
Ex② Check whether following L(G) is finite or not -

$S \rightarrow XS \mid b$
$X \rightarrow YZ$
$Y \rightarrow ab$
$Z \rightarrow XY$

Sol.- Step ① The given grammar is already completely reduced.

step ② we will draw a directed graph whose nodes will be S, X, Y, Z.

The graph is -



Here-

→ The directed graph contain cycles.

→ So, language of the given grammar is infinite.

③ CYK Algorithm:-

CYK algorithm is a membership algorithm of context free grammar.

→ It is used to decide whether a given string belongs to the language of grammar or not.

→ It is known as CYK Algorithm or "Cocke-Younger-Kasami Algorithm"

Note① CYK algorithm is applicable on CNF only
It is universal (applicable on all CNF)
Time Complexity - $O(n^3)$
Space Complexity - $O(n^2)$

★ Algorithm:-

```
Begin
For (i=1 to n do)
Vi1 {A | A→a is a production where ith symbol
      of X is a}
For (j=2 to n do)
for (i=1 to n-j+1 do)
Begin
  Vij = ϕ
 For K=1 to j-1 do
Vij = Vij ∪ {A | A→BC is a production where
      B is in Vik and C is in V(i+k)(j-k)
End
End
```

★ Deciding Membership using CYK algorithm:-

To decide the membership of any given string, we construct a triangular table where-

→ Each row of the table corresponds to one particular length of sub strings.
→ Bottom most row corresponds to strings of length -1.
→ Second row from bottom corresponds to strings of length-2
→ Third row from bottom corresponds to strings of length -3.
→ Top most row from bottom corresponds to the given string of length-n.

For a given string "x" of length 4 units, triangular table looks like-

X14			
X13	X23		
X12	X22	X32	
X11	X21	X31	X41

→ string of length-4
→ string of length-3
→ string of length-2
→ string of length-1

Ex- For the given grammar, check the acceptance of string w = baaba using CYK algorithm.

$S \rightarrow AB/BC$
$A \rightarrow BA/a$
$B \rightarrow CC/b$
$C \rightarrow AB/a$

Sol- For the triangular table :- X = baaba
Length of given string : $|X| = 5$
No. of sub strings possible = $(5 \times 6)/2 = 15$

For row - 1 :-

(I) $X_{11} = b$; $V_{11} = \{B\}$
(II) $X_{21} = a$; $V_{21} = \{A, C\}$
(III) $X_{31} = a$; $V_{31} = \{A, C\}$
(IV) $X_{41} = b$; $V_{41} = \{B\}$
(V) $X_{51} = a$; $V_{51} = \{A, C\}$

For row - 2 :-

Rule -
As per the algorithm, to find the value of V_{ij} from 2nd row on wards, we use formula -

$$V_{ij} = V_{ik} \cdot V_{(i+k)(j-k)}$$

where k varies from 1 to $j-1$

(I) $V_{12} = V_{11} \cdot V_{21}$
$V_{12} = \{B\}\{A, C\}$
$V_{12} = \{BA, BC\}$
$V_{12} = \{A, S\}$

(II) $V_{22} = V_{21} \cdot V_{31}$
$V_{22} = \{A, C\}\{A, C\}$
$V_{22} = \{AA, AC, CA, CC\}$
$V_{22} = \{CC\}$ {AA, AC and CA don't exists}
$V_{22} = \{B\}$

(III) $V_{32} = V_{31} \cdot V_{41}$
$V_{32} = \{A, C\}\{B\}$
$V_{32} = \{AB, CB\}$
$V_{32} = \{AB\}$ {CB doesn't exists}
$V_{32} = \{S, C\}$

(IV) $V_{42} = V_{41} \cdot V_{51}$
$V_{42} = \{B\}\{A, C\}$
$V_{42} = \{BA, BC\}$
$V_{42} = \{A, S\}$

For Row-3 :-

(I) $V_{13} = V_{11} \cdot V_{22} \cup V_{12} \cdot V_{31}$
$V_{13} = \{B\}\{B\} \cup \{A, S\}\{A, C\}$
$V_{13} = \{BB\} \cup \{AA, AC, SA, SC\}$
since BA, AA, AC, SA don't exist.
$V_{13} = \phi \cup \phi$
$V_{13} = \phi$

(II) $V_{23} = V_{21} \cdot V_{32} \cup V_{22} \cdot V_{41}$
$V_{23} = \{A, C\}\{S, C\} \cup \{B\}\{B\}$
$V_{23} = \{AS, AC, CS, CC\} \cup \{BB\}$
$V_{23} = \{CC\}$ {AS, AC, CS don't exists}
$V_{23} = B$

(iii) $V_{33} = V_{31} \cdot V_{42} \cup V_{32} \cdot V_{51}$

$V_{33} = \{A, C\}\{A, S\} \cup \{S, C\}\{A, C\}$

$V_{33} = \{AA, AS, CA, CS\} \cup \{SA, SC, CA, CC\}$

[since AA, AS, CA, CS, SA, SC & CA don't exists]

$V_{33} = \phi \cup \{CC\}$

$V_{33} = \phi \cup \{B\}$

$V_{33} = \{B\}$

For Row-4 :-

(i) $V_{14} = V_{11} \cdot V_{23} \cup V_{12} \cdot V_{32} \cup V_{13} \cdot V_{41}$

$= \{B\}\{B\} \cup \{A, S\}\{S, C\} \cup \{\phi, B\}$

$= \{BB\} \cup \{AS, AC, SS, SC\} \cup \{B\}$

[BB, AS, AC, SS, SC and B don't exist]

$V_{14} = \phi \cup \phi \cup \phi$

$V_{14} = \phi$

(ii) $V_{24} = V_{21} \cdot V_{33} \cup V_{22} \cdot V_{42} \cup V_{23} \cdot V_{51}$

$V_{24} = \{A, C\}\{B\} \cup \{B\}\{A, S\} \cup \{B\}\{A, C\}$

$V_{24} = \{AB, CB\} \cup \{BA, BS\} \cup \{BA, BC\}$

CB does not exists -

$V_{24} = \{AB\} \cup \{BA, BS\} \cup \{BA, BC\}$

$V_{24} = \{S, C\} \cup \{A\} \cup \{A, S\}$

$V_{24} = \{S, C, A\}$

For Row-5 :-

① $V_{15} = V_{11} \cdot V_{24} \cup V_{12} \cdot V_{33} \cup V_{13} \cdot V_{42} \cup V_{14} \cdot V_{51}$

$V_{15} = \{B\}\{S, C, A\} \cup \{A, S\}\{B\} \cup \{\phi\}\{A, S\} \cup \{\phi\}\{A, C\}$

$V_{15} = \{BS, BC, BA\} \cup \{AB, SB\} \cup \{A, S\} \cup \{A, C\}$

since BS, SB, S and C donot exists.-

$V_{15} = \{BC, BA\} \cup \{AB\} \cup \phi \cup \phi$

$V_{15} = \{S, A\} \cup \{S, C\} \cup \phi \cup \phi$

$V_{15} = \{S, A, C\}$

Now- We observe V_{15} contains the start symbol 'S'.
Thus string X_{15} = baaba is a member of the language of given grammar.

After filling the triangular table, it looks

{S,A,C}				
{φ}	{S,A,C}			
{φ}	{B}	{B}		
{S,A}	{B}	{S,C}	{S,A}	
{B}	{A,C}	{A,C}	{B}	{A,C}

✯ Results from Triangular Table -

* Result - 01 :-

→ There exists total 4 distinct sub strings which are member of the language of the given grammar. These substrings are - ba, ab, aaba, baaba.

→ This is because they contain start symbol in their respective cell.

✯ Result - 02 :-

→ Strings which can not be derived from any variable are - baa, baab

→ This is because they contain ϕ in their respective cell.

✯ Result - 03 :-

→ String which can be derived from variable B alone are - b, aa, aba, aab

→ This is because they contain variable B alone in their respective cell.