UNIT-III

Formal Grammar:

* Introduction
* classification of formal Grammar
    1. chomsky Hirearchy.
    2. Types.

* Introduction :—

Mathematically A formal Grammar is a tuple like

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

V = finite and non empty set of non-terminal symbols (or) Variables.

variables are represented by upper case letters

T = Finite and non empty set of Terminal symbols represented by lower case letters and some special symbols are there.

P = It is a set of production rules are of the form

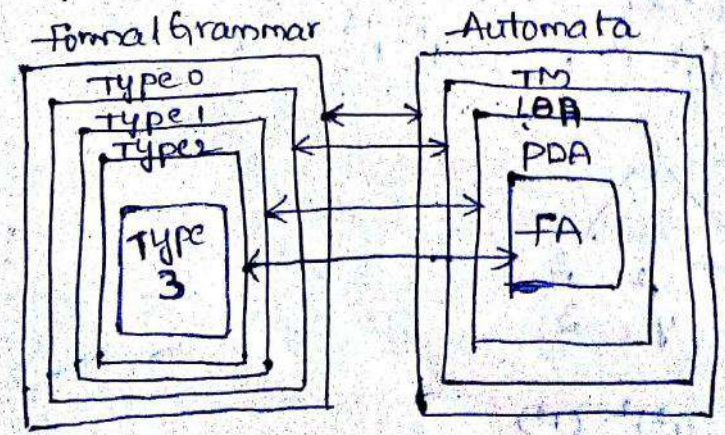
$$P \rightarrow \alpha \rightarrow \beta$$
$$\alpha \in V$$
$$\beta \in (V \cup T)^*$$

$S \rightarrow$ It is the starting symbol of the Grammar is always, a variable which is $S \in V$.

NOTE:— Grammar's are used to describe a language

* classification of Grammar :—

- using chomsky hierarchy.



Type 3 Grammar :—

* It is also called as Regular grammar.
* Type 3 Grammar is defined as $G = (V, T, P, S)$ where,

V → set of variables.
T → set of Terminals
P → set of production rules are of the form

$A \to Ba$ / $A \to a$ — According to left linear grammar

(or)

$A \to aB$ / $A \to a$ — According to Right linear grammar.

Ex :- $A \to aB$
$A \to Ba$
$A \to a$
$A \to \epsilon$

where.

$(A, B) \in V$

$a \in T^*$

* Type 3 Grammar is used to generating Regular language
* Regular languages are recognised (or) accepted by finite automata. i.e; NFA (or) DFA.

Type 2 Grammar :—

* It is also called as Context-free grammar.
* Ty context-free grammar is defined as $G = (V, T, P, S)$

where V → finite set of variables
T → finite set of Terminals
P → finite set of production rules are of the form

$\alpha \to \beta$

where $\alpha \in V$

$\beta \in (V \cup T)^*$

Ex :- $S \to aSa$
$S \to bSb$
$S \to ab$
$S \to \epsilon$

* Context-free grammars are used to generate Context-free language.

* context-free language recognised (or) Accepted by pushdown Automata.

Type 1 Grammar :—

* It is also called as context - Sensitive Grammar.
* A CSG is defined as $G=(V,T,P,S)$ where

V = finite set of variables

T = finite set of Terminals.

P = set of production rules are of the form $\alpha \rightarrow \beta$

where, $\alpha \in (VUT)^+$

$\beta \in (VUT)^*$

length of $|\alpha| \leq$ length of $|\beta|$

Ex:- $S \rightarrow aBb$

$bB \rightarrow aa$

$\beta \rightarrow b$

* CSG is used to generating Context - Sensitive language
* CSL recognised (or) Accepted by Linear Bounded Automata

Type 0 Grammar :—

* It is also called also Recursive - Grammar (or) Recursive Enumerable grammar. (or) phrase structured Grammar.
* mathematically Recursive grammar is defined as

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

$V \rightarrow$ finite set of variables

$T \rightarrow$ finite set of Terminals

$P \rightarrow$ set of production rules are of the form.

$\alpha \rightarrow \beta$

$\alpha \in (VUT)^{*+}$

$\beta \in (VUT)^*$

$|\alpha| \geq |\beta|$

Ex:- $S \rightarrow aAbB$

$aAbB \rightarrow aB$

$abB \rightarrow aA$

$A \rightarrow \epsilon$

* Recursive Grammars are used to generating recursive language (or) Recursive-Enumerable language (or) phrase structured language.

* Recursive languages are recognised are accepted by Turing machine.

Relationship b/w formal grammar and automata :—

1. Type 3 ⊆ Type 2 ⊆ Type 1 ⊆ Type 0
2. FA ⊆ PDA ⊆ LBA ⊆ TM

Context - Free Grammar :

* Introduction
* Design of CFL
* closure properties of CFL.

* Introduction) :—

Context-free Grammar is a Grammar which is defined by four tuples like $G=(V, T, P, S)$ where,

V – It is finite and non-empty set of non-terminal symbols (or) variables.

T – finite and non-Empty set of Terminal symbols.

P – finite and non-Empty set of production rules are of the form $\alpha \rightarrow \beta$

$\alpha \in V$

$\beta \in (V \cup T)^*$

Ex:- $S \rightarrow asa$

$S \rightarrow bSb$

$S \rightarrow aa|bb$

$S \rightarrow \epsilon$

S → It is starting symbol.

Context free -language :—

Let $G=(V, T, P, S)$ be a Context-free grammar. The CFG generating a language 'L' is called Context-free langua

*It is denoted by L(G)..

*Context-free languages are organized by PDA.

Design of CFL :—

1) Construct a CFL for the following set $\{\epsilon, a, aa, aaa, \cdots a^n\}$

Sol:- Given set $\{\epsilon, a, aa, aaa, aaaa, \cdots\cdots a^n\}$

minimum string = ϵ

Next minimum string = a

Maximum string = a^n

$S \rightarrow a^n$

$\Downarrow$

$a \cdot a^{n-1} \Rightarrow S \rightarrow aS$

$\Downarrow$

$a \cdot a \cdot a^{n-2}$ $\quad S \rightarrow \epsilon$

$\Downarrow$ $\quad S \rightarrow a$

$a \cdot a \cdot a \cdot a^{n-3}$

CFG:- $\quad S$

$S \rightarrow aS$

$S \rightarrow \epsilon$

$S \rightarrow a$

$L = \{a^n \mid n \geq 0\}$ //.

2) Construct a CFL for the following set $\{\epsilon, ab, aabb, \cdots\}$

Sol:- Minimum String = ϵ

Next minimum string = ab

Maximum string = $a^n b^n$

$S \rightarrow a^n b^n$

$S \rightarrow a\, a^{n-1} \cdot b^{n-1}\, b$

$S \rightarrow aa\, a^{n-2}\, b^{n-2}\, bb$

$\therefore S \rightarrow aSb$

$S \rightarrow \epsilon$

$S \rightarrow ab$

$\therefore$ CFG :- $S \rightarrow aSb$

$S \rightarrow \epsilon$

$S \rightarrow ab$

$\therefore L = \{a^n b^n \mid n \geq 0\}$

3) Construct a CFL for the following set {a, b, ab, aabb, aaabbb, ---}

Sol:- Minimum string = ab

Maximum string = $a^n b^n$

$S \rightarrow a^n b^n$

$\rightarrow a a^{n-1} b^{n-1} b$ $\Rightarrow$ $S \rightarrow aSb$

$\rightarrow aaa^{n-2} b^{n-2} bb$ $S \rightarrow ab$

$S \rightarrow b$.

$\therefore$ CFG $S \rightarrow asb$

$S \rightarrow a$

$S \rightarrow b$

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

4) Construct a CFG to generate the language $L = \{a^n b^{2n} / n \geq 1\}$

Sol:- Minimum string = abb

Maximum string = $a^n b^{2n}$

$S \rightarrow a^n b^{2n}$

$S \rightarrow a a^{n-1} b^{2n-2} bb$ $\Rightarrow$ $S \rightarrow aSbb$

$S \rightarrow abb$.

$\therefore$ CFG = $S \rightarrow asbb$

$S \rightarrow abb$.

5) Construct CFG for the following CFL

$L = \{ o^i 1^{i+1} \mid i \geq 0 \}$

Sol: $L = \{ o^i 1^{i+1} \mid i \geq 0 \}$

$= o^i 1^i . 1$

$A \rightarrow o^i 1^i$

$\rightarrow o o^{i-1} 1^{i-1} 1$

$S \rightarrow A1$

$\rightarrow oA1$

CFG: $S \rightarrow A1$

$A \rightarrow oA1$

$A \rightarrow \epsilon$

$A \rightarrow 01$

6) Construct a CFL from the following Language.

$L = \{ a^m b^m c^n \mid m, n \geq 0 \}$

Sol:-

$\underbrace{a^m b^m}_{A} \underbrace{c^n}_{B}$

$A \to a^m b^m$

$\to a\, a^{m-1} b^{m-1} b$

$A \to aAb$

$A \to \epsilon$

$A \to ab$

$B \to c^n$

$B \to c c^{n-1}$

$B \to cB$

$B \to \epsilon$

$B \to c$

CFG:- $S \to AB$

$A \to aAb$

$A \to \epsilon$

$A \to ab$

$B \to cB$

$B \to \epsilon$

$B \to c$

Closure properties of CFL:-

- content free languages are closed under union
- " " " " " " concatenation.
- " " " " " " Kleene closure
- " " " " " " Reversal
- content free languages are not closed under Complement
- " " " " " " Intersection
- " " " " " " difference.

** Derivation:-

* Introduction * Types of Derivation * Derivation tree

Derivation is a process of generating a string from a given grammar.

Derivation process can be represented graphically is called Derivation tree (or)

* left most derivation * Rightmost derivation.

Left most derivation:- With example

In this, we can replace a left most Variable to obtain the given input string.

Right most derivation:-

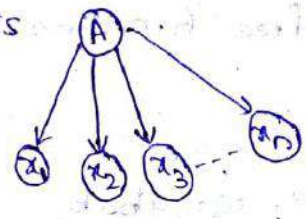
In this, we can replace a Right most variable to obtain the given input string.

Derivation Tree :—

Let $G = (V, T, P, S)$ be a CFG. Then there is a derivation tree for G. If and only if.

* the root node of the Tree is labelled with start Symbol of G
* All leaf nodes of Tree are labelled by Terminals (or) special Symbols of G.
* the interior nodes are labelled by variables of G.
* If any production rule in G. is are the form.

$A \rightarrow x_1 x_2 x_3 \cdots x_n$ then the derivation tree is



find the i) left most derivation
ii) Right most derivation
iii) parse tree for the i/p string id+id*id
from the following grammar. $E \rightarrow E+E$
$E \rightarrow E*E$
$E \rightarrow id$.

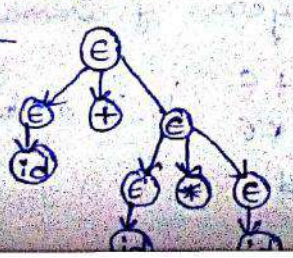
Sol:- the given grammar is $E \rightarrow E+E$
$E \rightarrow E*E$
$E \rightarrow id$.

Input string id+id*id.

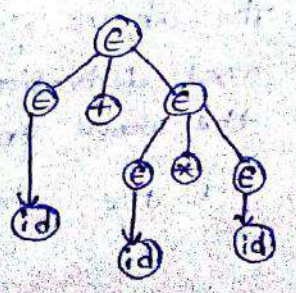
LMD:- $E \rightarrow \underline{E}+E$
$\rightarrow id+\underline{E}$
$\rightarrow id+\underline{E}*E$
$\rightarrow id+id*\underline{E}$
$\rightarrow id+id*id$.

RMD:- $E \rightarrow E+\underline{E}$
$\rightarrow E+E*\underline{E}$
$\rightarrow E+\underline{E}*id$
$\rightarrow \underline{E}+id*id$
$\rightarrow id+id*id$

Parse tree:-



Parse tree:-



Ambiguious Grammar:-

* A CFG $G=(V,T,P,S)$ which generates two (or) more parse trees for given i/p string is called Ambiguious grammar.

* that means an Ambegeous grammar has two or more left most derivations (or) right most derivation (or) parse tree.

Ex:- Prove that $S \to aSbS$, $S \to bSaS$, $S \to \epsilon$ is ambegeous for the i/p string abab

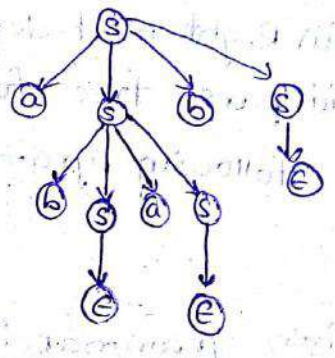
sol:- The given context free Grammar is $S \to aSbS$, $S \to bSaS$, $S \to \epsilon$.

the input string is $w = abab$

① LMD:

$S \to a\underline{S}bS$
$\Rightarrow ab\underline{S}aSbS$
$\to ab\epsilon aSbS$
$\to aba\underline{S}bS$
$\to aba\epsilon bS$
$\to abab\underline{S}$
$\to abab\epsilon$
$\to abab$

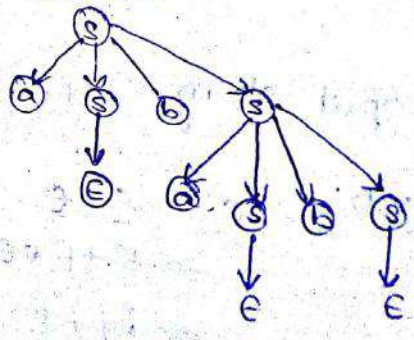
parsetree



② LMD:

$S \to a\underline{S}bS$
$\Rightarrow a\epsilon bS$
$\Rightarrow ab\underline{S}$
$\to aba\underline{S}bS$
$\to aba\epsilon bS$
$\to abab\underline{S}$
$\to abab\epsilon$
$\to abab$

Parsetree



∴ the above grammar generates two parsetrees (or) two left most derivation for the same i/p string $w=abab$. Hence the above grammar is ambegeous grammar.

2) P.T the grammar $E \to E+E$, $E \to E*E$, $E \to id$ is ambegeous for i/p string id+id*id.

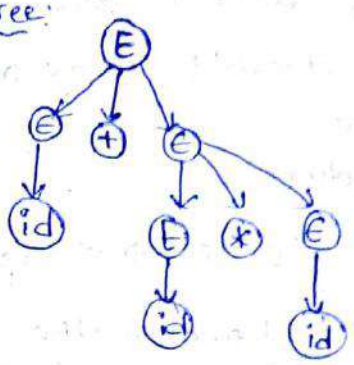
Sol:- The given context free grammar is $E \to E+E$
$E \to E*E$
$E \to id$

The input string is $w = id+id*id$.

① LMD:

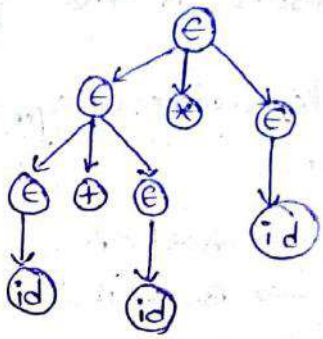
$E \to E+E$
$\to id+E$
$\to id+E*E$
$\to id+id*E$
$\to id+id*id.$

Parse tree:



② RMD:-

$E \to E*E$
$\to E+E*E$
$\to id+E*E$
$\to id+id*E$
$\to id+id*id.$



** (7m) Simplification of CFG:

* Introduction
* Methods
  1. Elimination of useless symbols.
  2. Elimination of ϵ-productions
  3. Elimination of unit productions.

Introduction :-

It's means minimizing the no. of productions in the given CFG. that is reducing size of CFG. size of CFG is equal to no. of productions.

Methods :- $S \to AB$
$A \to a$
$A \to aA$
$B \to SB$

Elimination of useless symbols :-

useful symbol :- A variable is said to be useful if and only if

* It generates a terminal string.
* It is used in derivation of a string at least one time

useless symbol :-

* A variable is said to be useless if and only if.
  * It doesn't generates a terminal string.
* It doesn't used in derivation of a string at least one time.

Procedure :-

step 1 :- Determine useless symbols in the grammar.

step 2 :- Remove the productions which contains useless symbols in the grammar.

Ex :- eliminate useless symbols from the following grammar.

$S \rightarrow AB \mid CA$
$B \rightarrow BC \mid AB$
$A \rightarrow a$
$C \rightarrow aB \mid b$

sol :- The given CFG is

$S \rightarrow AB$
$S \rightarrow CA$
$B \rightarrow BC$
$B \rightarrow AB$
$A \rightarrow a$
$C \rightarrow aB$
$C \rightarrow b$

In the given grammar 'B' doesn't generating a terminal string.

$S \rightarrow \underline{A}B$
$S \rightarrow a\underline{B}$
$\rightarrow a\underline{B}C$
$\rightarrow a\underline{A}BC$
$\rightarrow aaBC$
$\rightarrow aaBb$
$\rightarrow aa\underline{A}Bb$
$\rightarrow aaaBb$
:

∴ so, 'B' is useless symbol.

so, we can eliminate the productions which contains 'B'.

∴ The reduced CFG is

$S \rightarrow CA \mid C \rightarrow b.$
$A \rightarrow a$

2) elimination of ∈-production :-

∈-production :- A production is of the form $A \to \epsilon$ is called ∈-production (or) NULL production.

procedure :-

step 1 :- If the grammar contains $A \to \epsilon$ then replace 'A' with ∈ in the remaining productions.

step 2 :- Remove $A \to \epsilon$ from the grammar.

ex :- Remove ∈-productions from the following grammar

$A \to 0B1 \mid 1B1$

$B \to 0B \mid 1B \mid \epsilon$

sol :- the given CFG is $A \to 0B1$

$A \to 1B1$

$B \to 0B$

$B \to 1B$

$B \to \epsilon$.

$A \to 0B1$
$\to 0\epsilon1$
$\to 01$

$\therefore A \to 0B1$
$A \to 01$

$B \to 0B$
$\to 0\epsilon$
$\to 0$

$\therefore B \to 0B$
$B \to 0$

$B \to 1B1$
$\to 1\epsilon1$
$\to 11$

$\therefore A \to 1B1$
$A \to 11$

$B \to 1B$
$\to 1\epsilon$
$\to 1$

$\therefore B \to 1B$
$B \to 1$

After eliminating $B \to \epsilon$ the resultant CFG is

$A \to 0B1$
$A \to 01$
$A \to 1B1$
$A \to 11$
$B \to 0B$
$B \to 0$
$B \to 1B$
$B \to 1$

14m

Normal forms :-

* Introduction
* Types of Normal forms
    1. chomsky Normal form (CNF)
    2. Greibach Normal form (GNF)

Introduction :-

In CFG each production of the form $\alpha \rightarrow \beta$ where $\alpha \in V$ that means β contains any no. of non-terminal symbols and any no. of terminal symbols. But, we need to have a grammar in specific form i.e; we can decide the no. of non-terminals and terminals on RHS of the grammar. This can be implemented by using "normalization of CFG".

Normalization :-

The process of Arranging the grammar with fixed no. of

non-terminals and terminals on R.H.S of CFG is called normalization.

normal forms are classified into two types

i) chomsky normal form.

ii) Greibach normal form

chomsky normal form :—

It is defined as $\alpha \rightarrow \beta$

> non-terminal → Non-terminal. Non-terminal.
>
> (or)
>
> Non-terminal → Terminal.

conversion of CFG to CNF :—

procedure :-

step 1 :- simplify the CFG

step 2 :- convert the simplified CFG to CNF.

Ex:- convert the following CFG into chomsky normal form.

$S \rightarrow aaaaS$

$S \rightarrow aaaa$

Sol:- The given grammar is $S \rightarrow aaaaS$

$S \rightarrow aaaa$

Consider a non-terminal $A \Rightarrow$ that derives terminal a.

$\therefore$ The production rule is $A \rightarrow a$. is in CNF

$S \rightarrow aaaaS$

$S \rightarrow AAAAS$ can be replaced by P_1

$S \rightarrow AP_1$ is in CNF.

$P_1 \rightarrow AAAS$ can be replaced by P_2

$P_1 \rightarrow AP_2$ is in CNF

$P_2 \rightarrow AAS$ can be replaced by P_3

$P_2 \rightarrow AP_3$ is in CNF

$P_3 \rightarrow AS$ is in CNF

$S \rightarrow aaaa$

$S \rightarrow AAAA$ can be replaced by P_4

$S \to AP_4$ is in CNF

$P_4 \to A[AA]$ can be replaced by P_5.

$P_4 \to AP_5$ is in CNF

$P_5 \to AA$ is in CNF.

The resultant Grammar CNF is

$S \to AP_1$ $\quad P_5 \to AA$

$S \to AP_4$ $\quad A \to a$

$P_1 \to AP_2$

$P_2 \to AP_3$

$P_3 \to AS$

$P_4 \to AP_5$

2) Convert the given CFG to CNF. $S \to aSa$

$S \to bSb$

$S \to a$

$S \to b$.

Sol:- The given grammar is $S \to aSa$

$S \to bSb$.

$S \to a$

$S \to b$

It is already in simplified form.

Consider a non-terminal A that derives a terminal a and the non-Terminal B. that derives the terminal b.

∴ The production rules is $A \to a$ are in CNF.

$B \to b$.

(i) $S \to aSa$.

$S \to A[SA]$ can be replaced by P_1

$S \to AP_1$ is in CNF.

$P_1 \to SA$ is in CNF.

(ii) $S \to bSb$

$S \to B[SB]$ can be Replaced by P_2

$S \to BP_2$ is in CNF

$P_2 \to SB$ is in CNF

(iii) $S \to a$ is in CNF

$S \to b$ is in CNF.

∴ The resultant grammar in CNF is

$S \to AP_1$

$S \to BP_2$

$S \to a$

$S \to b$

$P_1 \to SA$

$P_2 \to SB$

$A \to a$

$B \to b$.

Greibach Normal form (GNF) :-

GNF is defined as

Non-terminal $\longrightarrow$ Terminal . any no. of nonterminals

Non-terminal $\longrightarrow$ Terminal.

Lemma 1 :

Let CFG be $G = (V, T, P, S)$ and there is a production rule $A \to \alpha B$ and $B \to \beta_1 | \beta_2 | \beta_3 | \ldots\ldots | \beta_n$ then add the new production rule to GNF $A \to \alpha\beta_1 | \alpha\beta_2 | \alpha\beta_3 | \ldots | \alpha\beta_n$ to GNF.

$\therefore$ B is replaced by $B \to \beta_1 | \beta_2 | \ldots | \beta_n$

Lemma 2 :

Let CFG be $G = (V, T, P, S)$ and there is production rule

$A \to A\alpha_1 | A\alpha_2 | \ldots | A\alpha_n | \beta_1 | \beta_2 | \ldots\ldots | \beta_n$ then the production rules are added to GNF.

$A \to \beta_1 | \beta_2 | \beta_3 | \ldots | \beta_n$

$A \to \beta_1 Z | \beta_2 Z | \beta_3 Z | \ldots | \beta_n Z$

$Z \to \alpha_1 | \alpha_2 | \alpha_3 | \ldots | \alpha_n$

$Z \to \alpha_1 Z | \alpha_2 Z | \alpha_3 Z | \ldots | \alpha_n Z$

Converting CFG into GNF :-

Procedure :-

step 1 :- Simplify the CFG.

step 2 :- Converting simplified CFG into GNF.

Ex :- Convert the given CFG to GNF. $S \to ABA$

$A \to aA | \epsilon$

$B \to bB | \epsilon$

Sol :- The Given CFG is $S \to ABA$

$A \to aA$

$A \to \epsilon$

$B \to bB$

$B \to \epsilon$

Simplified of given CFG :-

(a) elimination of ϵ-productions :-

$A \to \epsilon$ $B \to \epsilon$

(1) $S \to ABA$
$S \to \epsilon BA$
$S \to BA$

(2) $S \to ABA$
$S \to AB\epsilon$
$S \to AB$

(3) $S \to ABA$
$S \to A\epsilon A$
$S \to AA$

(4) $S \to ABA$
$S \to \epsilon\epsilon A$
$S \to A$

(5) $S \to ABA$
$S \to \epsilon B\epsilon$
$S \to B$

$A \to aA$ $B \to bB$
$A \to a\epsilon$ $B \to b\epsilon$
$A \to a$ $B \to b$

∴ After eliminating $A \to \epsilon$, $B \to \epsilon$ from the grammar the resultant grammar is

$S \to ABA$ $A \to aA$
$S \to BA$ $A \to a$
$S \to AB$ $B \to bB$
$S \to AA$ $B \to b$
$S \to A$
$S \to B$

Elimination of unit productions :-

The above grammar has two unit productions like

$S \to A$ × $S \to B$ ×

$S \to aA$ $S \to bB$ [∵ since $A \to aA$ $B \to bB$
$S \to a$ $S \to b$ $A \to a$ $B \to b$]

∴ After elimination unit productions $S \to A$, $S \to B$ from the grammar. The resultant grammar is

$S \to ABA$ $A \to aA$
$S \to BA$ $A \to a$
$S \to AB$ $B \to bB$
$S \to AA$ $B \to b$
$S \to aA$
$S \to a$
$S \to bB$
$S \to b$

there is no useless production.

∴ The simplified CFG is

$S \to ABA$
$S \to BA$
$S \to AB$
$S \to AA$
$S \to aA$
$S \to a$
$S \to bB$
$S \to b$

$A \to aA$
$A \to a$
$B \to bB$
$B \to b$

Converting simplified CFG to GNF :

i) $S \to ABA$
$S \to aABA$ ✓
$S \to aBA$ ✓

ii) $S \to BA$
$S \to bBA$ ✓
$S \to bA$ ✓

iii) $S \to AB$
$S \to aAB$
$S \to aB$ ✓

iv) $S \to AA$
$S \to aAA$
$S \to aA$

v) $S \to aA$ ✓
$S \to a$ ✓

vi) $S \to bB$ ✓
$S \to b$ ✓

$A \to aA$ ✓
$A \to a$ ✓

$B \to bB$ ✓
$B \to b$ ✓

∴ The resultant grammar is in GNF. is

$S \to aABA \mid aBA \mid bBA \mid bA \mid aAB \mid aB \mid aAA \mid aA \mid bB \mid a \mid b$
$A \to aA \mid a$
$B \to bB \mid b$.

(2) Convert the following CFG into GNF $S \to AA \mid 0$
$A \to SS \mid 1$

Sol: Given Grammar $S \to AA$
$S \to 0$
$A \to SS$
$A \to 1$

The simplified CFG is $S \to AA$
$S \to 0$
$A \to SS$
$A \to 1$

① $S \to AA|0$

$S \to SSA|0$

$S \to 0$

$S \to 0Z$

$Z \to SA$

$Z \to SAZ$

$Z \to SA$

$Z \to 0A$

$Z \to 0ZA$

$Z \to 1AA$

$Z \to 0A$

② $S \to AA|0$

$S \to 1A|0$

$S \to 1A$

$S \to 0$

③ $A \to SS$

$A \to 0S$

$A \to 0ZS$

$A \to 1AS$

$A \to 0S$

$Z \to SAZ$

$Z \to 0AZ$

$Z \to 0ZAZ$

$Z \to 1AAZ$

$Z \to 0AZ$

∴ The resultant grammar is

$S \to 0|0Z|1A$

$Z \to 0A|0ZA|1AA|0A|0AZ|0ZAZ|1AAZ|$

$A \to 0S|0ZS|1AS|1$

③ convert the given CFG to GNF $S \to CA$

$A \to a$

$C \to aB|b$

⇒ Given CFG is not a simplified Grammar

After eliminating the useless symbols the resultant CFG is. $S \to CA$

$A \to a$

$C \to b$

By Applying Lemma 1 $S \to CA$

$S \to bA$

∴ The resultant GNF is $S \to bA$

$A \to a$

$C \to b$

④ Convert the given CFG to GNF $S \to SS$

$S \to OS1|01$

The given CFG is a simplified CFG

The resultant grammar is $S \to SS$

$S \to OS1$

$S \to 01$

Replaced 0 by A, 1 by B

Then productions are $A \to 0$

$B \to 1$

S → S S

S → A S B

S → A B

Applying Lemma ①

① S → S S

S → A S B S

S → 0 S B S

② S → S S

S → A B S

S → 0 B S

② S → A S B

S → 0 S B

S → A B

S → 0 B

The resultant grammar GNF is

S → 0 S B S | 0 B S | 0 S B | 0 B

A → 0

B → 1

Pumping Lemma for CFL:—

pumping Lemma is used for proving the given language is CFL (or) not.

Lemma :- let 'L' be any CFL, then there is a constant 'n' which depends only a part 'L' such that there exist a string.

$w = uvxyz$ such that 1. $|vy| \geq 1$

2. $|vxy| \leq n$

3. for $\forall i \geq 0$ $uv^i xy^i z$ is in 'L'

Then 'L' is said to be CFL. otherwise it is not a CFL.

① Prove that $L = \{a^n b^n c^n | n \geq 0\}$ is not a CFL.

The given language $L = \{a^n b^n c^n | n \geq 0\}$.

$L = \{\epsilon, abc, aabbcc, ----\}$

consider a constant n and the string $w = a^n b^n c^n$

consider a string $w \in L$

$w = abc$ for $n = 1$

$|w| = 3n$

for $i = 1$ $w = abc$

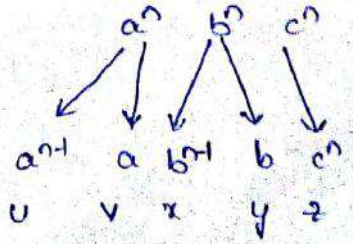
for $i = 2$

$w = uv^i xy^i z$

$w = uv^2 xy^2 z$

$w = a^{n-1} a^2 b^{n-1} b^2 c^n$

$w = a^{n+1} b^{n+1} c^n \notin L$



∴ The given language is not a CFL.

② show that the language $L = \{ ss^T / s \in \{a,b\}^* \}$

Given language $L = \{ ss^T / s \in \{a,b\}^* \}$

$L = \{ \epsilon,$

UNIT-IV

PUSH DOWN AUTOMATA

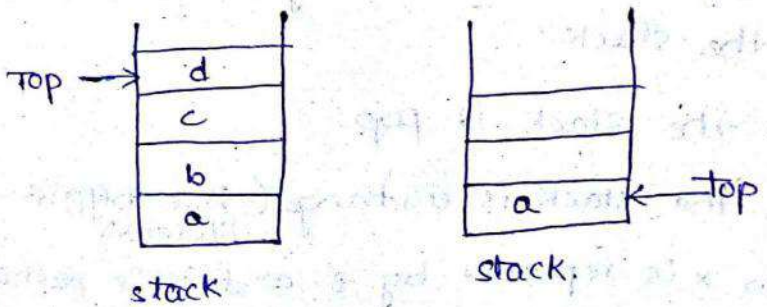
* Introduction * Basic model [Formal definition] * Graphical Notation
* Instantaneous Description (ID) * Acceptance of PDA.

Introduction:-

* A PDA is a way to implement a CFG in a similar way we can design FA for Regular Grammar.
* PDA is more powerful than finite state machine.
* FSM has a very limited memory. But a PDA has more memory.
* PDA = FSM + stack
* A stack is a way we arrange elements one on the top of stack.
* A stack does two basic operations.
  - i) PUSH :- A new element is added at the top of the stack.
  - ii) POP :- The top element of the stack is read and remove.

Ex:
push(a)
push(b)
push(c)
push(d)

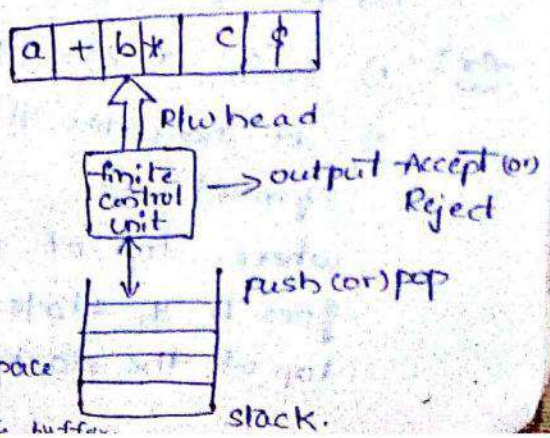
pop()
pop()
pop()



* **Basic Model of PDA :-**

PDA has three Components.

i) input tape
ii) finite control unit.
iii) stack



* A stack with infinite size.
* It has unlimited amount of storage space
* Used to store data and remove the data temporarily which is read by FA from input buffer

Formal definition:-

- Mathematically a PDA is defined with 7 tuples like

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

$Q \rightarrow$ finite and non-empty set of states

$\Sigma \rightarrow$ finite and non-empty set of input symbols

$\Gamma \rightarrow$ finite and non-empty set of stack symbols.

$\delta \rightarrow$ It is a transition function which is defined as

δ:

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

$$Q \times \Sigma^* \times \Gamma^* \longrightarrow Q \times \Gamma^*$$

where 'δ' takes three tuples as i/p like $\delta(q, a, x)$

where i) q is a state in Q.

ii) a is either an i/p symbol in Σ (or) a is also belongs ϵ.

iii) x is a stack symbol i.e; member of Γ

iv) The o/p of δ is finite set of pairs like (p, γ)

where, p: It is a new state.

γ: It is a set of stack symbols that replace 'x' at the top of the stack.

Ex:- 1) If $\gamma = \epsilon$ then the stack is pop.

2) If $\gamma = x$ then the stack is unchange (since bypass operation)

3) If $\gamma = yz$ then x is replaced by z and 'y' is pushed on to the stack.

Ex:- 1) $\delta(q_0, a, z) = (q_1, yz)$

⇒ It indicates that from state q_0, reading i/p symbol 'a' where top of the stack z. then the finite control goes to q_1 state and adding the element 'y' to the top of the stack.

2) $\delta(q_1, a, z) = (q_2, \epsilon)$

⇒ It indicates that 'z' is removed from the stack and state is changed from q_1 to q_2.

3) $\delta(q_1, a, z) = (q_2, z)$

⇒ It indicates that on reading symbol 'a' state is changing from q_1 to q_2 and there is no change in the stack (bypass operation)

q_0 → It is the initial state.

$q_0 \in Q$

z_0 → It is the start stack symbol.

$z_0 \in T$

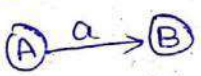
F → It is the set of final (or) accepting state and

$(F \subset Q)$.

Graphical representation :-

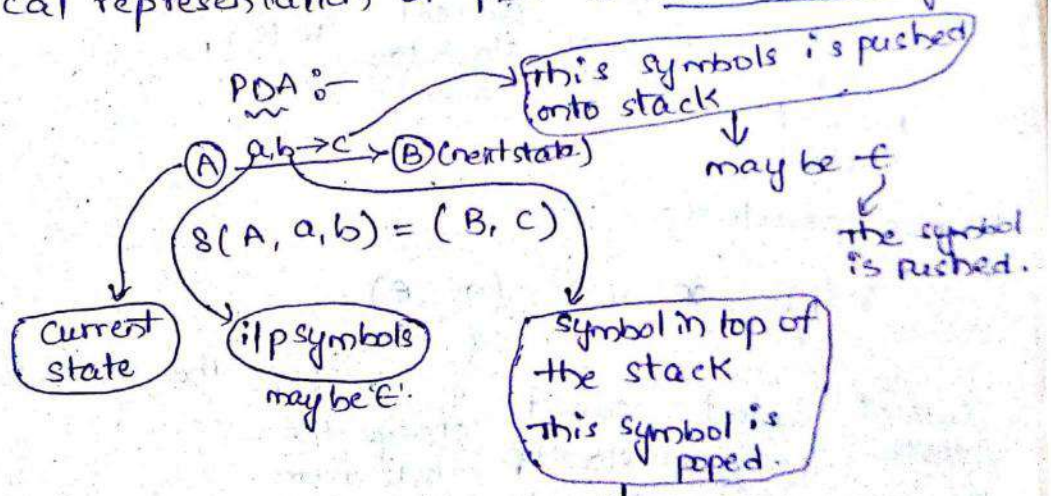
The Graphical representation of PDA is Transition diagram

FA :-



$\delta(A, a) = B$

PDA :-



Instantaneous description :-

It is used to describe the configuration of PDA at given Instance.

ID remembers the state and stack content.

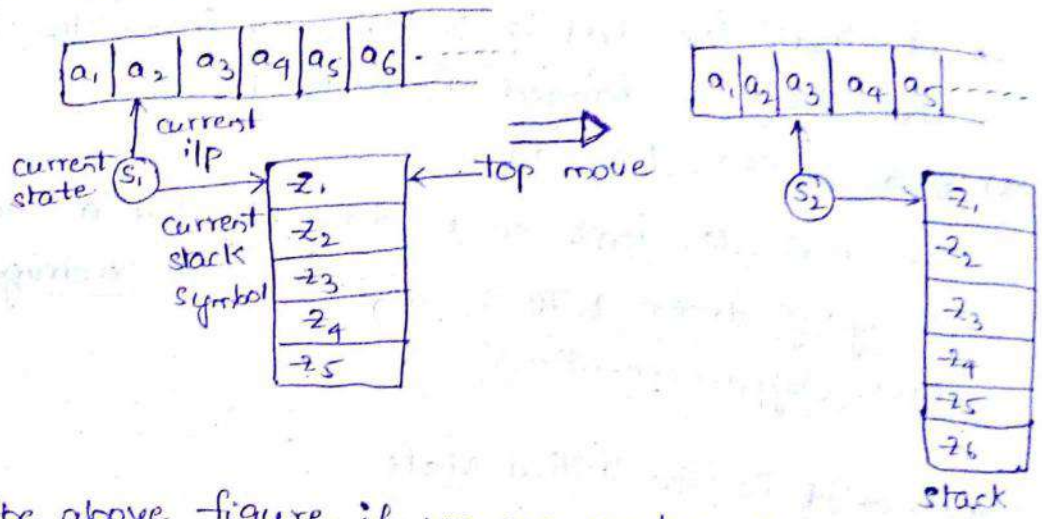
It was defined by Triple (q, w, γ) where

q → is a state.

w → input symbols of string

γ → is a string of stack symbols.

Example :- $\delta(P, a, z_0) = (q, \beta)$



From the above figure, if we are reading the current i/p symbol 'a_2' at current state 's_1' and current stack symbol 'z_1' then after a move we will reach to state s_2. and there will be some new symbol on the top of the stack. This description can be represented as.

1) push operation :—

$\delta(q_0, a_1, z_0) = (q_1, az_0)$ → push 'a' onto the stack.

- q_0 : current state (or) present state
- a_1 : current i/p symbol
- z_0 : current stack top symbol.
- q_1 : change the state from q_0 to q_1

2) pop operation :—

$\delta(q_0, x, y) = (q_1, \epsilon)$

- q_0 : current state
- x : current i/p symbol
- y : current stack top symbol.
- q_1 : change the state from q_0 to q_1
- ϵ : pop the stack (or) removing the stack.

Acceptance of PDA :—

There are two ways to accept a language by PDA. They are i) Accepted by empty stack.
ii) Accepted by final state.

Accepted by empty stack :—

The given language Accepted by empty stack to be defined as $L(M) = \{w \mid \delta(q_0, w, z_0) \overset{*}{\Rightarrow} (P, \epsilon, \epsilon)$ for some p in $Q\}$

that is, if stack becomes empty after scanning entire string then it is accepted by PDA - otherwise, not accepted.

Accepted by final string:-

The given language accepted by final state to be defined as

$L(M) = \{ w \mid \delta(q_0, w, z_0) \xrightarrow{*} (p, \epsilon, f)$ for some $p \in F$ and $f \in \Gamma\}$

that is, eventhough stack is not empty, after scanning i/p string. if the finite control reaches to the final state then it is accepted. otherwise, not accepted.

Design of PDA :-

Types of PDA :-

i) Deterministic PDA :- if all derivations in the design has to give only single move

ii) Non Deterministic PDA :- if derivation generates more than one move in the designing of a particular task.

1) Design a PDA that accepts equal no. of A's and B's.

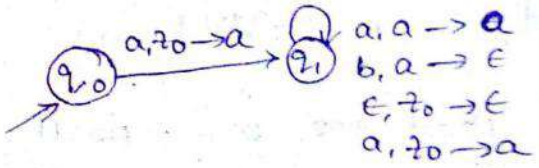
Sol:- δ: $\delta(q_0, a, z_0) = (q_1, a z_0)$

$\delta(q_1, a, a) = (q_1, aa)$

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

$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$

$\delta(q_1, a, z_0) = (q_1, a z_0)$



∴ The PDA machine for the above language is defined as

$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ where $Q = \{q_0, q_1\}$

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

$\Gamma = \{z_0\}$

δ :

$q_0 = \{q_0\}$

$z_0 = \{z_0\}$

$F = \{\ \}$

Read A's → push operation, Read B's → pop operation

(1) Consider a string $w = \{abab\}$ Read a's

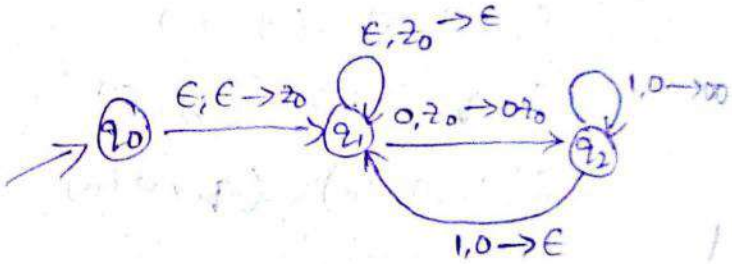
$\delta(q_0, abab, z_0) = \delta(q_1, bab, a z_0)$

③ Design a PDA for the language $L = \{0^n 1^{2n} / n \geq 1\}$

sol :- $L = \{0^n 1^{2n} / n \geq 1\}$

Read one 0 → push

Read two 1's → pop



$\delta(q_0, \epsilon, \epsilon) = (q_1, z_0)$

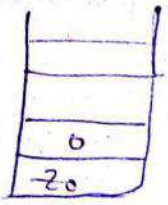
$\delta(q_1, 0, z_0) = (q_2, 0z_0)$

$\delta(q_2, 0, 0) = (q_2, 00)$

$\delta(q_2, 1, 0) = (q_2, 00)$

$\delta(q_2, 1, 0) = (q_1, \epsilon)$

$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon, \epsilon)$



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

$\delta(q_1, 1, 0) = (q_1, 0z_0)$

④ Consider the string $w = \{001111\}$

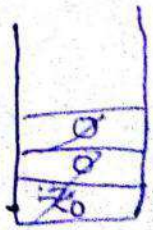
$\delta(q_1, 001111, z_0) = \delta(q_2, 01111, 0z_0)$

$= \delta(q_2, 1111, 00)$

$= \delta(q_2, 111, 00)$

$= \delta(q_2, 11, 0z_0)$

$= \delta(q_1, 1, 0z_0)$



stack

$= \delta(q_1, \epsilon, z_0)$

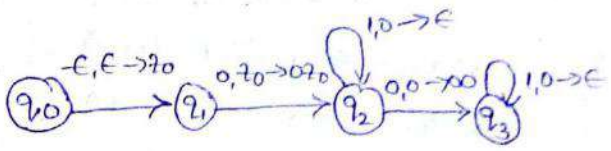
$= \delta(q_1, \epsilon, \epsilon)$

//

Design a PDA for the language $L = \{0^n 1^n \mid n \geq 1\}$ (a b)

Read 0's → push

Read 1's → pop



$\delta(q_0, \epsilon, \epsilon) = (q_1, z_0)$

$\delta(q_1, 0, z_0) = (q_2, 0z_0)$

$\delta(q_2, 0, 0) = (q_2, 00)$

$\delta(q_2, 1, 0) = (q_3, \epsilon)$

$\delta(q_3, 1, 0) = (q_3, \epsilon)$

$\delta(q_3, \epsilon, z_0) = (q_3, \epsilon, \epsilon)$

* Design a PDA for the language $L = \{ww^R \mid w \in (a+b)^*\}$

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

sol) i) $L = \{ww^R \mid w \in (a+b)^*\}$

In this language contains palindrome string. i.e;

if $w = ab$, $w^R = ba$ then $ww^R = abba$ is a palindrome.

* we can read no. of a's and b's and pushed them into stack until we can reach the mid position of i/p string.
* In the mid position we can't read any i/p and can't push onto stack.
* After mid position when we read a (or) b then pop them from the stack. This process is repeated until stack is empty.

$\delta(q_0, \epsilon, \epsilon) = (q_1, z_0)$

$\delta(q_1, a, z_0) = (q_1, az_0)$

$\delta(q_1, b, z_0) = (q_1, bz_0)$

$\delta(q_1, \epsilon, \epsilon) = (q_2, z_0)$

$\delta(q_2, a, a) = (q_3, \epsilon)$

$\delta(q_2, b, b) = (q_3, \epsilon)$

$\delta(q_3, \epsilon, z_0) = (q_4, \epsilon, \epsilon)$

ii) $L = \{ \omega c \omega^R \mid \omega = (a+b)^* \}$

$\omega = ab$

$\omega^R = ba$

$\omega c \omega^R = abcba$

$\delta(q_0, \epsilon, \epsilon) = (q_1, Z_0)$

$\delta(q_1, a, Z) = (q_1, aZ_0)$

$\delta(q_1, b, Z_0) = (q_1, bZ_0)$

$\delta(q_1, a, a) = (q_1, aa)$

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

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

$\delta(q_1, b, b) = (q_1, bb)$

$\delta(q_1, c, Z_0) = (q_2, Z_0)$

$\delta(q_1, c, a) = (q_2, a)$

$\delta(q_1, c, b) = (q_2, b)$

$\delta(q_2, a, a) = (q_3, \epsilon)$

$\delta(q_3, b, b) = (q_3, \epsilon)$

$\delta(q_3, \epsilon, Z_0) = (q_4, \epsilon, \epsilon)$

Deterministic pushDown Automata:-

A DPDA is 7 Tuple like $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

where Q is finite and non empty set of states

Σ is finite and nonempty set of ilp Alphabet

Γ is finite set of stack symbols

δ is a mapping function used for mapping (or) moving from current state to next state. is defined as $\delta(q_0, x, Z_0) = (q, x\beta)$ where

q_0 is current state

x is current ilp symbol

Z_0 is current stack symbol

q is next state

$x\beta$ shows top of the stack

if δ denotes a unique transition for each ilp then PDA is said to be deterministic PDA

Ex:- i) $L = \{ a^n b^n \mid n \geq 1 \}$

2) $L = \{ \omega c \omega^R \mid \omega = (a+b)^* \}$

Nondeterministic PDA:-

It is a 7 tuple like $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where

Q is finite and non empty set of states

Σ is finite and non empty set of ilp Alphabet

Γ is finite set of stack symbols.

δ is a mapping function used for moving from current state to next state and is defined as $\delta(q_0, x, z_0) = (q, \alpha\beta)$

q_0 is current state
x is current i/p symbol
z_0 is stack symbol
q is next state
$\alpha\beta$ is top of the stack

if δ denotes more than one transition for a particular i/p symbol then the PDA is said to be non-deterministic PDA.

Ex:- $L = \{ ww^R / w = (a+b)^* \}$

Context free grammar and push down automata:-

Conversion of CFG to PDA.

conversion of PDA to CFG.

i) Conversion of CFG to PDA:-

* for constructing a PDA from given CFG it is necessary to convert this CFG to some Normal form like GNF.

* For converting given CFG to PDA, By this method the necessary condition is that the first symbol on RHS of production rule must be a terminal symbol. This rule that can be used to obtain PDA from CFG.

Algorithm:-

Rule 1:- for non-terminal symbols, add following rule

$\delta(q, \epsilon, A) = (q, \alpha)$ where the production rule is $A \to \alpha$.

Rule 2:- for each terminal symbols, add following rule

$\delta(q, a, a) = (q, \epsilon)$ for every terminal symbol 'a' in given CFG.

Ex:- construct a PDA for the given CFG $S \to 0BB$
$B \to 0S$
$B \to 1S$
$B \to 0$

Sol:- The given CFG $G = (V, T, P, S)$ where V = non-terminals $\{S, B\}$

$T = \{0, 1\}$

P :-> $S \to OBB$

$B \to 0S$

$B \to 1S$

$B \to 0$

$S = \{s\}$

Rule 1 :- $A \to \alpha$

$\delta(q, \epsilon, A) = (q, \alpha)$

$S \to OBB$

$\delta(q, \epsilon, S) = (q, OBB)$

$B \to 0S$

$\delta(q, \epsilon, B) = (q, 0S)$

$B \to 1S$

$\delta(q, \epsilon, B) = (q, 1S)$

$B \to 0$

$\delta(q, \epsilon, B) = (q, 0)$

Rule :-2

Terminals

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

$T = \{0, 1\}$

$\delta(q, 0, 0) = (q, \epsilon)$

$\delta(q, 1, 1) = (q, \epsilon)$

∴ The corresponding PDA for the given CFG is defined as

$M = (Q, \Sigma, \tau, \delta, q_0, z_0, F)$

$Q = \{q\}$

$\Sigma = \{0, 1\}$

$\tau = \{S, B, 0, 1\}$

δ = it is a transition symbol. defined as

$\delta(q, \epsilon, S) = (q, OBB)$

$\delta(q, \epsilon, B) = (q, 0S)$

$\delta(q, \epsilon, B) = (q, 1S)$

$\delta(q, \epsilon, B) = (q, 0)$

$\delta(q, 0, 0) = (q, \epsilon)$

$\delta(q, 1, 1) = (q, \epsilon)$

$q_0 = \{q\}$

$z_0 = \{z_0\}$

$F = \{\}$

2) construct a PDA for the following CFG $S \to 0S1$, $S \to A$, $A \to 1A0 | S | \epsilon$.

sol:- The given CFG is $S \to 0S1$, $S \to A$, $A \to 1AD$, $A \to S$, $A \to \epsilon$.

elimination of ϵ-production:-

$A \to \epsilon$ ×
$S \to A$
$S \to \epsilon$ ×
$S \to 0S1$
$S \to 0\epsilon 1$
$S \to 01$

$A \to 1AD$
$A \to 1\epsilon 0$
$A \to 10$
$A \to S$
$A \to \epsilon$ ×

∴ $S \to 0S1 | 01$
$S \to A$
$A \to 1A0 | 10$
$A \to S$.

elimination of unit productions:-

$S \to A$ ×
$S \to 1A0 | 10$

$A \to S$ ×
$A \to 0S1 | 01$

∴ The resultant CFG is.

$S \to 1A0$
$S \to 10$
$A \to 0S1$
$A \to 01$

∴ The Simplified CFG is.

$S \to 1AD$
$S \to 1D$
$A \to 0S1 | 1A0 | 10$
$A \to 01$
$S \to 0S1 | 01$

Method-1
$P \to 1$
$Q \to 0$

$S \to 1A0$ $S \to 1D$ $A \to 0S1$ $A \to 1D$ $S \to 0S1$
$S \to 1AQ$ ✓ $S \to 1Q$ ✓ $A \to 0SP$ ✓ $A \to 1Q$ ✓ $S \to 0SP$ ✓

$A \to 1AD$ $A \to 01$ $S \to 01$
$A \to 1AQ$ ✓ $A \to 0P$ ✓ $S \to 0P$ ✓

∴ the simplified CFG in GNF is $S \to 1AQ$, $S \to 1Q$, $S \to 0SP$, $S \to 0P$, $A \to 0SP$, $A \to 1Q$, $A \to 1AQ$, $A \to 0P$, $P \to 1$, $Q \to 0$

Rule:-1 ∴ The PDA is

$\to \alpha$. $S \to 1AQ$.
$\delta(q_1, \epsilon, S) = (q_1, 1AQ)$

$S \to 1Q$
$\delta(q_1, \epsilon, S) = (q_1, 1Q)$

$S \to 0SP$
$\delta(q_1, \epsilon, S) = (q_1, 0SP)$

$S \to 0P$
$\delta(q_1, \epsilon, S) = (q_1, 0P)$

$A \to 0SP$
$\delta(q_1, \epsilon, A) = (q_1, 0SP)$

$A \to 1Q$
$\delta(q_1, \epsilon, A) = (q_1, 1Q)$

$A \to 1AQ$

$\delta(q, \epsilon, A) = (q, 1AQ)$

$A \to 0P$

$\delta(q, \epsilon, A) = (q, 0P)$

$P \to 1$

$\delta(q, \epsilon, P) = (q, 1)$

$Q \to 0$

$\delta(q, \epsilon, Q) = (q, 0)$

Rule 2:- Terminals.

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

$T = \{0, 1\}$

$\delta(q, 0, 0) = (q, \epsilon)$

$\delta(q, 1, 1) = (q, \epsilon)$

method :- 2

The Given CFG is $S \to 0S1$

$S \to A$

$A \to 1A0$

$A \to S$

$A \to \epsilon$

The resultant PDA is $S \to 0S1$

$\delta(q, \epsilon, S) = (q, 0S1)$

$S \to A$

$\delta(q, \epsilon, S) = (q, A)$

$A \to 1A0$

$\delta(q, \epsilon, A) = (q, 1A0)$

$A \to S$

$\delta(q, \epsilon, A) = \delta(q, S)$

$A \to \epsilon$

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

Construct PDA for the following CFG $S \to aABB \mid aAA$

$A \to aBB \mid a$

$B \to bBB \mid A$

sol:- The Given CFG is $S \to aABB$

$S \to aAA$

$A \to aBB$

$A \to a$

$B \to bBB$

$B \to A$

Elimination of unit production :-

$B \to Ax$

$B \to aBB$

$B \to a$

∴ After eliminating unit production $B \to A$, the resultant CFG in GNF is $S \to aABB$

$S \to aAA$

$A \to aBB$

$A \to a$

$B \to bBB$

$B \to aBB$

$B \to a$

∴ The PDA is:

$S \to aABB$

$\delta(q, \epsilon, S) = (q, aABB)$

$S \to aAA$

$\delta(q, \epsilon, S) = (q, aAA)$

$A \to aBB$

$\delta(q, \epsilon, A) = (q, aBB)$

$A \to a$

$\delta(q, \epsilon, A) = (q, a)$

$B \to bBB$

$\delta(q, \epsilon, B) = (q, bBB)$

$B \to aBB$

$\delta(q, \epsilon, B) = (q, aBB)$

$B \to a$

$\delta(q, \epsilon, B) = (q, a)$

Conversion of PDA to CFG :—

* If $M = (Q, \Sigma, T, \delta, q_0, Z_0, F)$ is a PDA. then their exists CFG 'G' which is accepted by PDA (M).

Let 'G' be a CFG which is generated by PDA. The 'G' can be defined as $G = (V, T, P, S)$ where 'S' is the start symbol and the set of non-terminals $V = \{S, q, q', Z_0\}$ where $S, q, q' \in Q$ and $Z_0 \in T$.

Now, we get set of production rules using the following algorithm.

Algorithm :—

Rule 1 :— The start symbol production rule can be $S \to [q, Z_0, q']$

where q indicates present state

q' indicates next state.

Z_0 is the stack symbol.

Rule 2 :— If there exists a move of PDA as then the production rule can be return as $\delta(q, a, Z_0) = (q', \epsilon)$

$[q, Z_0, q'] \to a$

3) If there exists a move of PDA as $\delta(q_1, a, Z_0) = (q', Z_1 Z_2 Z_3 \cdots)$ then the production rules can be written as

$$[q, Z_0, q'] \rightarrow a[q_1, Z_1, q_1][q_1, Z_2, q_2][q_2, Z_3, q_3] \cdots [q_{n-1} Z_n q']$$

Ex:- construct a CFG from the following PDA $M = (\{q_0, q_1\}, \{0, 1\}, \{S, A\}, \delta, q_0, S, \phi)$ and

δ :

$\delta(q_0, 1, S) = (q_0, AS)$

$\delta(q_0, \epsilon, S) = (q_0, \epsilon)$

$\delta(q_0, 1, A) = (q_0, AA)$

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

$\delta(q_1, 1, A) = (q_1, \epsilon)$

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

Sol:- Let we will construct a CFG $G = (V, T, P, S)$ where

$T = \{0, 1\}$

$V = \{S \cup [q_0, S, q_0], [q_0, S, q_1], [q_1, S, q_0], [q_1, S, q_1], [q_0, A, q_0], [q_0, A, q_1], [q_1, A, q_0], [q_1, A, q_1]\}$

Now, Let us build the production rules as.

using rule ① the production rules for start symbol is

P_1: $S \rightarrow [q_0, S, q_0]$

P_2: $S \rightarrow [q_0, S, q_1]$

using Rule ③ of the Algorithm for the $\delta(q_0, 1, S) = (q_0, AS)$

$q_0 < q_0 q_0$ P_3: $[q_0, S, q_0] \rightarrow 1[q_0, A, q_0][q_0, S, q_0]$

$q_0 < q_0 q_1$ P_4: $[q_0, S, q_0] \rightarrow 1[q_0, A, q_1][q_1, S, q_0]$

P_5: $[q_0, S, q_1] \rightarrow 1[q_0, A, q_0][q_0, S, q_1]$

P_6: $[q_0, S, q_1] \rightarrow 1[q_0, A, q_1][q_1, S, q_1]$

Now, for $\delta(q_0, \epsilon, S) = (q_0, \epsilon)$ using Rule ② of Algorithm we get.

P_7: $[q_0, S, q_0] \rightarrow \epsilon$

P_8: ~~$[q_0, S, A] \rightarrow q_0 A$~~

Now for $\delta(q_0, 1, A) = (q_0, AA)$ using Rule ③ of Algorithm.

P_8: $[q_0, A, q_0] \rightarrow 1[q_0, A, q_0][q_0, A, q_0]$

Now for $\delta(q_0, 0, A) = (q_1, A)$ using Rule ② of Algorithm

$P_9: [q_0, A, q_0] \rightarrow 0 [q_0, A, q_1] [q_1, A, q_0]$

$P_{10}: [q_0, A, q_1] \rightarrow 1 [q_0, A, q_0] [q_0, A, q_1]$

$P_{11}: [q_0, A, q_1] \rightarrow 1 [q_0, A, q_1] [q_1, A, q_1]$

Now, for $\delta(q_0, 0, A) = (q_1, A)$ using

$q_0 \begin{cases} q_0 \\ q_1 \end{cases}$ $q_1 \begin{cases} q_0 \\ q_1 \end{cases}$

$P_{12}: [q_0, A, q_0] \rightarrow 0 [q_1, A, q_0]$

$P_{13}: [q_0, A, q_1] \rightarrow 0 [q_1, A, q_1]$

Now, for $\delta(q_1, 1, A) = (q_1, \epsilon)$

$P_{14}: [q_1, A, q_1] \rightarrow 1$

Now, for $\delta(q_1, 0, s) = (q_0, s)$

$q_1 \begin{cases} q_0 \\ q_1 \end{cases}$ $q_0 \begin{cases} q_0 \\ q_1 \end{cases}$

$P_{15}: [q_1, s, q_0] \rightarrow 0 [q_0, s, q_0]$

$P_{16}: [q_1, s, q_1] \rightarrow 0 [q_0, s, q_1]$

PDA with two stacks :—

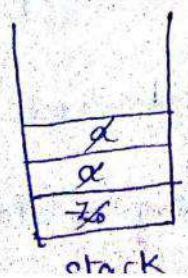
ⓐ PDA with one stack :

① $L = \{a^n b^n \mid n \geq 1\}$

Consider the string $w = aabb$

read a's → push

read b's → pop



stack

a a b b $

when stack is empty then the string aabb is accepted.

② $L = \{a^n b^n c^n / n \geq 1\}$

consider string $w = aabbcc$

read a's → PUSH

read b's → POP

read c's → No change

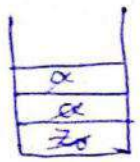
stack.

a a b b c c $

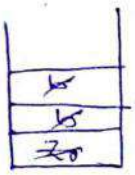
when read c there is no change in stack without completion of reading string 'w' the stack is empty. so, string is not accepted.

ⓑ PDA with two stacks :—

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



stack 1



stack 2

w = aabbcc $

read a's → PUSH (on $stack_1$)

read b's → PUSH (on $stack_2$)

read c's → pop (a from $stack_1$ and 'b' from $stack_2$)

when two stacks are empty then string 'w' is accepted.

∴ The PDA with two stacks is more powerful than a ~~stack~~ PDA with one stack.

FA + 0-stack = NFA or DFA

FA + 1-stack = PDA.

FA + 2-stack = PDA with two stacks.

- Applications of PDA :—

* used for deriving a string from the grammar.
* used for designing top-down parser and bottom-up parser in compiler design.
* It works on regular grammar and Content-free grammars
* It accepts regular language and CFL.
* It has remembering capability by maintaining a stack.
* It is more powerful than FA.

3/3/18

UNIT-5

TURING MACHINE

Turing machine contains 7 tuples $TM = (Q, \Sigma, \Gamma, \delta, q_0, F, B)$

where Q = Set of states

Σ = set of input symbols

Γ = set of input tape symbols

δ = Transition function

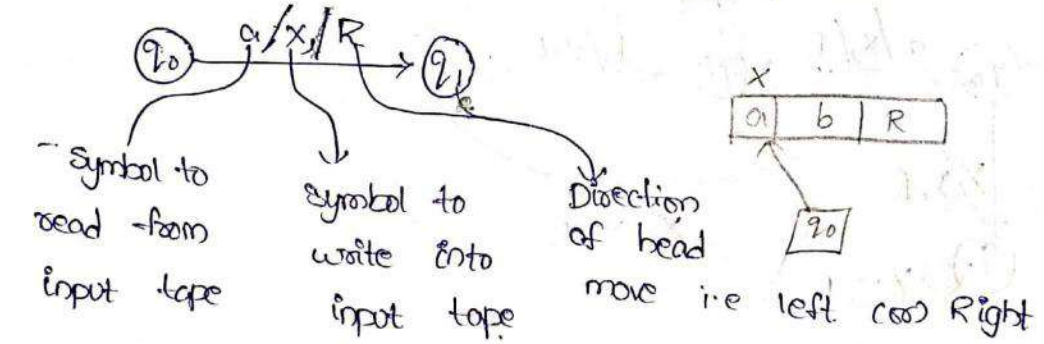
q_0 = initial state

F = final state

B = Blank symbol.

where δ can be defined as $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times (L/R)$

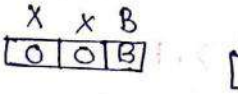
The Transition Diagram:- The transition function can be represented in the form of graphical notation.

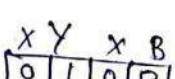


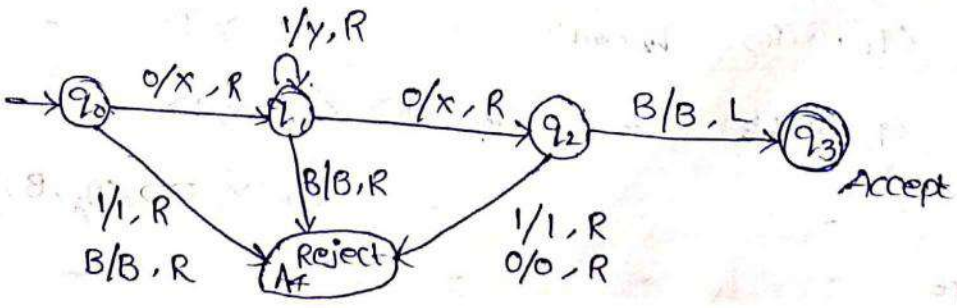
1) Design a turing machine for $L = 01^*0$

$L = 01^*0$

$L = \{00, 010, 0110, 01110, \ldots\}$





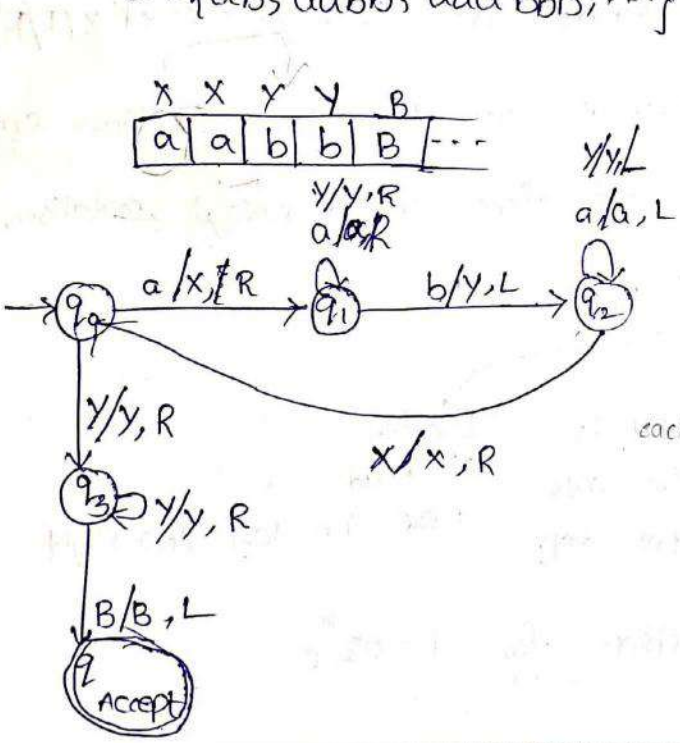


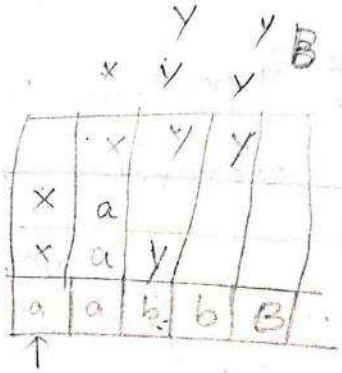
Tape symbol / State	0	1	X	Y	B
q_0	$\langle q_1, X, R\rangle$	$\langle q_4, 1, R\rangle$	-	-	$\langle q_4, B, R\rangle$
q_1	$\langle q_2, X, R\rangle$	$\langle q_1, Y, R\rangle$	-	-	$\langle q_4, B, R\rangle$
q_2	$\langle q_4, 0, R\rangle$	$\langle q_4, 1, R\rangle$	-	-	$\langle q_3, B, L\rangle$
q_3	-	-	-	-	-
q_4	-	-	-	-	-

* **

2) Design a Turing Machine for language $L = \{a^n b^n / n \geq 1\}$

$L = \{ab, aabb, aaabbb, \cdots\}$





each 'a' is compared to 'b'

Tape symbol / State	a	b	X	Y	B
$\rightarrow q_0$	$\langle q_1, X, R\rangle$	—	—	$\langle q_3, Y, R\rangle$	—
q_1	$\langle q_1, a, R\rangle$	$\langle q_2, Y, L\rangle$	-	$\langle q_1, Y, R\rangle$	-
q_2	$\langle q_2, a, L\rangle$	—	$\langle q_0, X, R\rangle$	$\langle q_2, Y, L\rangle$	-
q_3	—	-	—	$\langle q_3, Y, R\rangle$	$\langle q_A, B, L\rangle$
* q_{Accept}	-	-	-	-	-

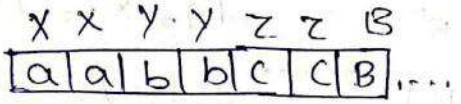
~~Transition functions~~

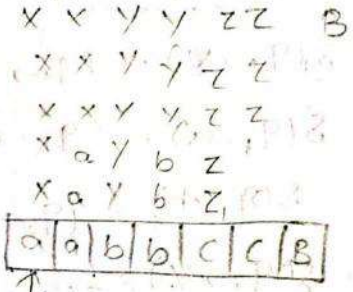
8

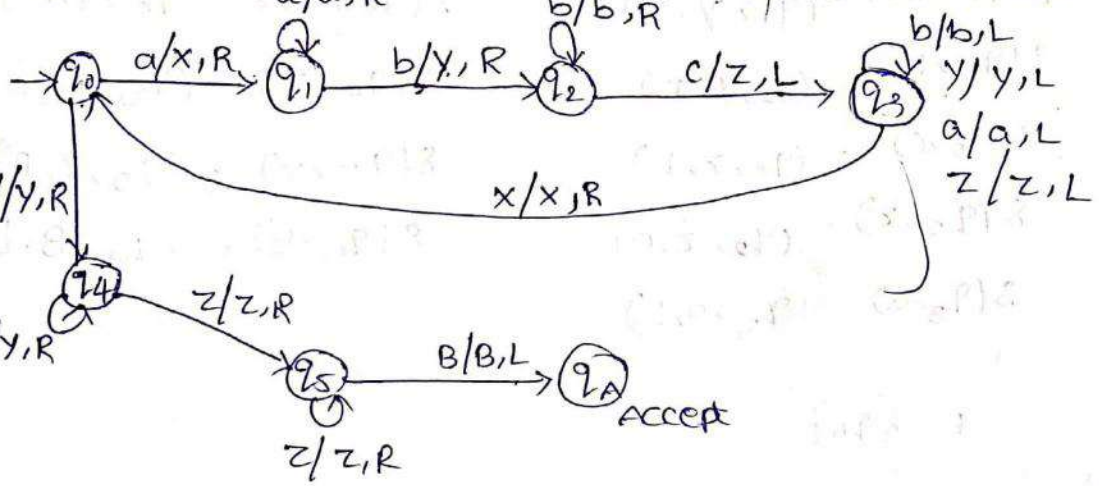
5) Design Turing machine for $L = \{a^n b^n c^n / n \geq 1\}$

$L = \{abc, aabbcc, aaabbbccc, \ldots\}$

$TM = \{Q, \Sigma, \delta, \Gamma, F, q_0, B\}$







Tape symbol / State	a	b	c	X	Y	Z	B
→ q_0	$\langle q_1, X, R\rangle$	–	–	–	$\langle q_4, Y, R\rangle$	–	–
q_1	$\langle q_1, a, R\rangle$	$\langle q_2, Y, R\rangle$	–	–	$\langle q_1, Y, R\rangle$	–	–
q_2	–	$\langle q_2, b, R\rangle$	$\langle q_3, Z, L\rangle$	–	–	$\langle q_2, Z, R\rangle$	–
q_3	$\langle q_3, a, L\rangle$	$\langle q_3, b, L\rangle$	–	$\langle q_0, X, R\rangle$	$\langle q_3, Y, L\rangle$	$\langle q_3, Z, L\rangle$	–
q_4	–	–	–	–	$\langle q_4, Y, R\rangle$	$\langle q_5, Z, R\rangle$	–
q_5	–	–	–	–	–	$\langle q_5, Z, R\rangle$	$\langle q_A, B, L\rangle$
q_A	–	–	–	–	–	–	–

TM: $M = \{Q, \Sigma, \Gamma, \delta, q_0, B, F\}$

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

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

$\Gamma = \{a, b, c, x, y, z, B\}$

$\delta(q_0, a) = (q_1, x, R)$

$\delta(q_0, y) = (q_4, y, R)$

$\delta(q_1, a) = (q_1, a, R)$

$\delta(q_1, b) = (q_2, y, R)$

$\delta(q_1, y) = (q_1, y, R)$

$\delta(q_2, b) = (q_2, b, R)$

$\delta(q_2, c) = (q_3, z, L)$

$\delta(q_2, z) = (q_2, z, R)$

$\delta(q_3, a) = (q_3, a, L)$

$\delta(q_3, b) = (q_3, b, L)$

$\delta(q_3, x) = (q_0, x, R)$

$\delta(q_3, y) = (q_3, y, L)$

$\delta(q_3, z) = (q_3, z, L)$

$\delta(q_4, y) = (q_4, y, R)$

$\delta(q_4, z) = (q_5, z, R)$

$\delta(q_5, z) = (q_5, z, R)$

$\delta(q_5, B) = (q_A, B, L)$

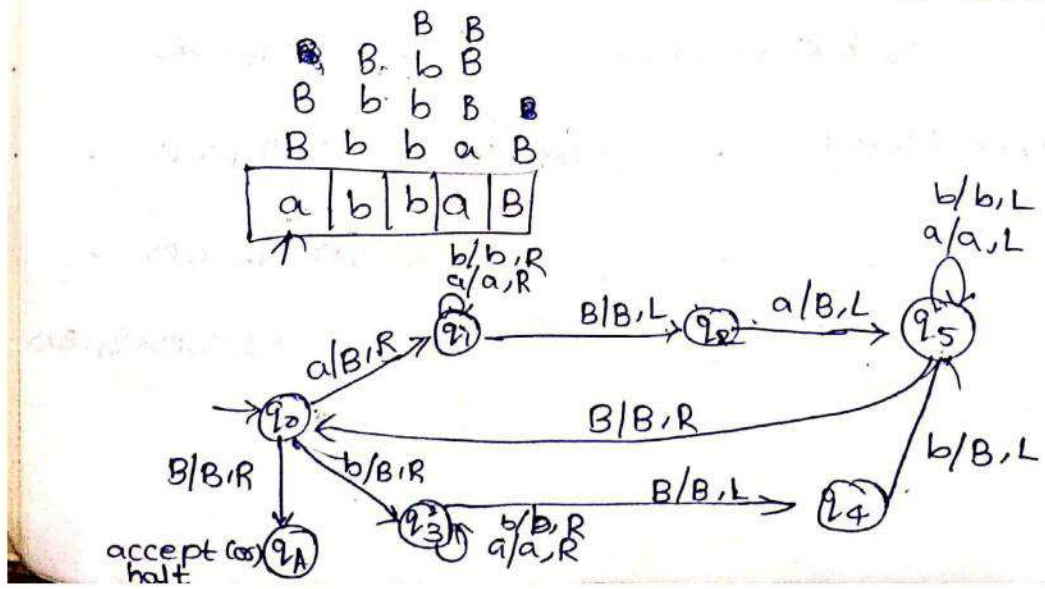
$F = \{q_A\}$

4) Design Turing Machine for $L = \{ww^R / w \in (a, b)^*\}$

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

It is a even length palindrome

$L = \{aa, bb, abba, baab, abbbba, abaaba, \ldots\}$



Tape Symbols / State	a	b	B
$\rightarrow q_0$	$\langle q_1, B, R\rangle$	$\langle q_3, B, R\rangle$	$\langle q_A, B, R\rangle$
q_1	$\langle q_1, a, R\rangle$	$\langle q_1, b, R\rangle$	$\langle q_2, B, L\rangle$
q_2	$\langle q_5, B, L\rangle$	-	-
q_3	$\langle q_3, a, R\rangle$	$\langle q_3, b, R\rangle$	$\langle q_4, B, L\rangle$
q_4	-	$\langle q_5, B, L\rangle$	-
q_5	$\langle q_5, a, L\rangle$	$\langle q_5, b, L\rangle$	$\langle q_0, B, R\rangle$
* q_A	-	-	-

(1) abba

q_0 a b b a B

$\vdash$ B q_1 b b a B

$\vdash$ B b q_1 b a B

$\vdash$ B b b q_1 a B

$\vdash$ B b b a q_1 B

$\vdash$ B b b q_2 a B

$\vdash$ B b q_5 b B B

$\vdash$ B q_5 b b B B

$\vdash$ B q_5 B b b B B

$\vdash$ B q_0 b b B B

$\vdash$ B B q_3 b B B

$\vdash$ B B b q_3 B B

$\vdash$ B B q_4 b B B

$\vdash$ B q_5 B B B B

$\vdash$ B B q_0 B B B

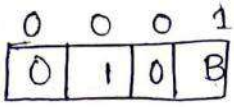
$\vdash$ B B B q_A B B

Accept

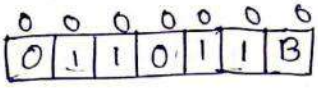
5) Design turing Machine for 'parity counter' that outputs '0' (or) '1' depending on w

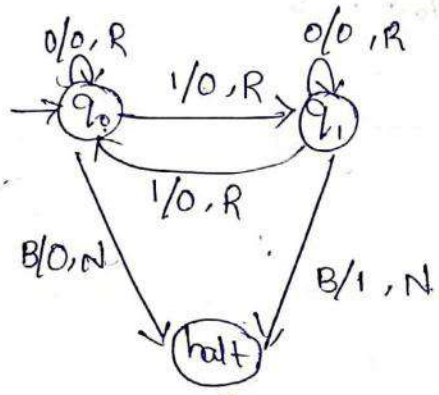
1's odd - 1

1's even - 0



Transition Table





odd 1's 010

$\vdash q_0 010\ B$

$\vdash 0 q_0 10\ B$

$\vdash 00 q_0 0\ B$

$\vdash 000 q_1 B$

$\vdash 0001$ halt

even 1's 1010

$\vdash q_0 1010\ B$

$\vdash 0 q_1 010 B$

$\vdash 00 q_1 10 B$

$\vdash 000 q_0 0B$

$\vdash 0000 q_0 B$

$\vdash 00000$ halt

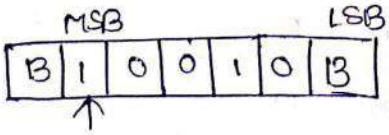
Design TM for 2's complement

I/P 10010

1's 01101

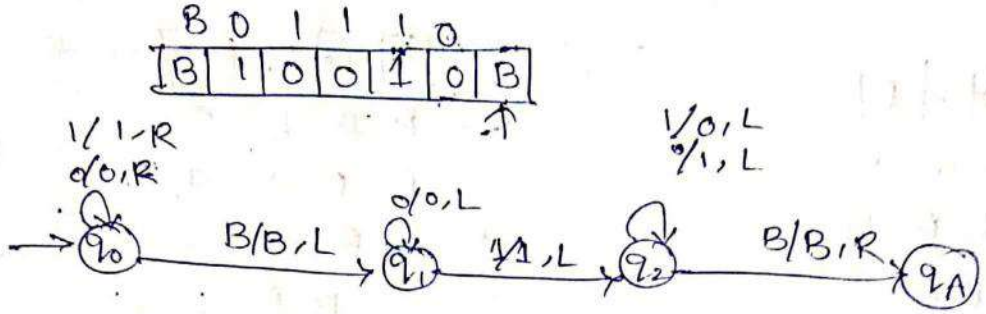
+1 1

2's 01110



accept
halt

First of all we have to move LSB then upto



ID:- B1 0 0 1 0 B

$\vdash B q_0 1 0 0 1 0 B$

$\vdash B 1 q_0 0 0 1 0 B$

$\vdash B 1 0 q_0 0 1 0 B$

$\vdash B 1 0 0 q_0 1 0 B$

$\vdash B 1 0 0 1 q_0 0 B$

$\vdash B 1 0 0 1 0 q_0 B$

$\vdash B 1 0 0 1 q_1 0 B$

$\vdash B 1 0 0 q_1 1 0 B$

$\vdash B 1 0 q_2 0 1 0 B$

$\vdash B 1 q_2 0 1 1 0 B$

$\vdash B q_2 1 1 1 1 0 B$

$\vdash B B 0 1 1 1 0 B$

$\vdash q_2 B 0 1 1 1 0 B$

$\vdash B q_2 0 1 1 1 0 B$

* Addition of two integers. Design TM)*

Design turing Machine for to accept set of all the palindromes.

Sol:

a	b	a	B

B b a B

B b B B

B B B

halt

b	a	b	B

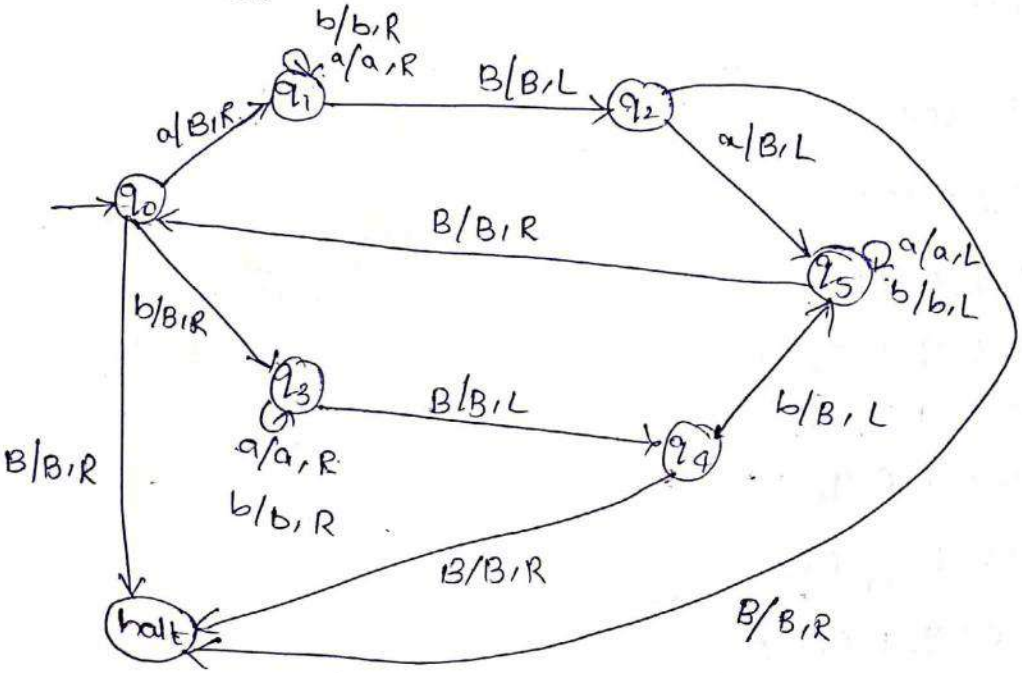
B a b B

B a B B

B B B

B B

halt



bab

$\vdash q_0 bab B$

$\vdash B q_3 ab B$

$\vdash B a q_3 b B$

$\vdash B a b q_3 B$

$\vdash B a q_4 b B$

$\vdash B q_5 a B B$

$\vdash q_5 B a B B$

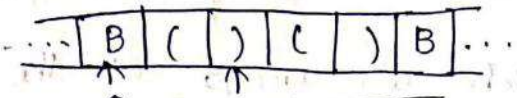
$\vdash B q_0 a b B$

$\vdash B B q_1 B B$

$\vdash B q_2 B B B$

$\vdash B B q_A B B$

Design Turing Machine for parenthesis checking



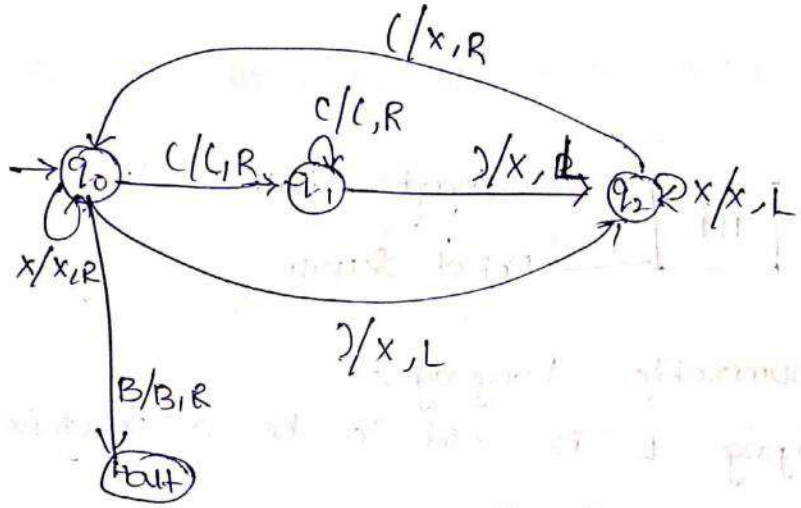
B (X ( ) B

B X X ( ) B

B X X (X B

B X X X X B

↑ B



⊢ q_0 (( )) B

(q_1 ( )) B

((q_1)) B

((q_2 X) B

(q_2 (X) B

(X q_0 X) B

(X X q_0) B

(X q_0 X X B

(q_2 X X X B

q_2 (X X X B

X q_0 X X X B

X X q_0 X X B

X X X q_0 X B

X X X X q_0 B

X X X X B H

Types of Grammars - Chomsky Hierarchy:

Linguist Noam Chomsky defined a hierarchy of languages in terms of complexity. This four level hierarchy, called the Chomsky hierarchy, corresponds to four classes of machines.

The Chomsky hierarchy classifies grammars according to form of their productions into the following four levels.

(1) Type 0 grammars - unrestricted grammars
(2) Type 1 grammars - Context sensitive grammars
(3) Type 2 grammars - Context free grammars
(4) Type 3 grammars - regular grammars.

Type 0
Type 1
Type 2
Type 3

(1) Type-0 grammars - Unrestricted Grammars (URG)

These grammars include all formal grammars. In URGs, all the productions are of the form $\alpha \rightarrow \beta$, where α and β may have any number of terminals and non-terminals. ie, no restrictions on either side of productions.

Every grammar is included in it if it has atleast one non-terminal on the left hand side.

Ex:- $aA \rightarrow abCB$
$aA \rightarrow bAA$
$bA \rightarrow a$
$S \rightarrow aAb \mid \epsilon$

They generate exactly all languages that can be recognized by a turing machine. The language that is recognized by a Turing machine is defined as set of all the strings on which it halts. These Languages

are also known as the recursively enumerable languages. (2)

(2) Type 1 grammar – Context Sensitive Grammars: (CSG)

These grammars define the context-sensitive languages. In Context sensitive grammar, all the productions of the form $\alpha \to \beta$, where length of α is less than or equal to the length β. i.e $|\alpha| \leq |\beta|$, α and β are may have any number of terminals and non-terminals. These languages are exactly all the languages that can be recognized by linear bound automata.

(2)

Ex:- $|\alpha| \leq |\beta|$ $\alpha \to \beta$

$aAbcD \to abcDbcD$

(3) Type 2 Grammar – Context-free Grammar (CFG)

These grammars define the context-free languages. These are defined by rules of the form $\alpha \to \beta$ with $|\alpha| \leq |\beta|$ where $|\alpha| = 1$ and α is non-terminal and β is a string of terminals and non-terminals. we can replace α by β regardless of where it appears. Hence the name context free grammar. These languages are exactly those languages that can be recognized by a pushdown automaton. CFG's Context free languages defines the syntax of all programming languages.

Ex:- (i) $S \to aS|Sa|a$

(ii) $S \to aAA|bBB|\epsilon$.

(4) Type 3 grammars - regular grammars:

These grammars generate the regular languages. Such a grammar restricts its rules to a single non-terminal on the LHS. the RHS consists of either a single terminal or string of terminals with single non-terminal on left or right end.

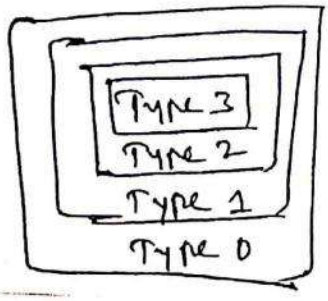
$\alpha \to \beta$, $\alpha = \{V\}$
$\beta = V\{T\}^* \cup T^*$

Ex: $A \to aA \mid a$ — right linear grammar
$A \to Aa \mid a$ — left linear grammar.

* Every regular language is context free, every context-free language is context-sensitive and every context-sensitive language is recursively enumerable.

Table: Chomsky's Hierarchy

Grammar	Language	Automaton	production rules
Type 0	Recursively enumerable	Turing machine	$\alpha \to \beta$ no restrictions on β, α should have atleast one non-terminal
Type 1	Context-sensitive	Linear bounded Automata	$\alpha \to \beta$ $\lvert\alpha\rvert \le \lvert\beta\rvert$
Type 2	Context-free	Push down automata	$\alpha \to \beta$ $\lvert\alpha\rvert = 1$
Type 3	Regular	Finite state automata	$\alpha \to \beta$, $\alpha = \{V\}$ $\beta = \{V\}T^* \cup T^*$ $= T^*\{V\}$ $= T^*$



Chomsky hierarchy of grammars

(3)

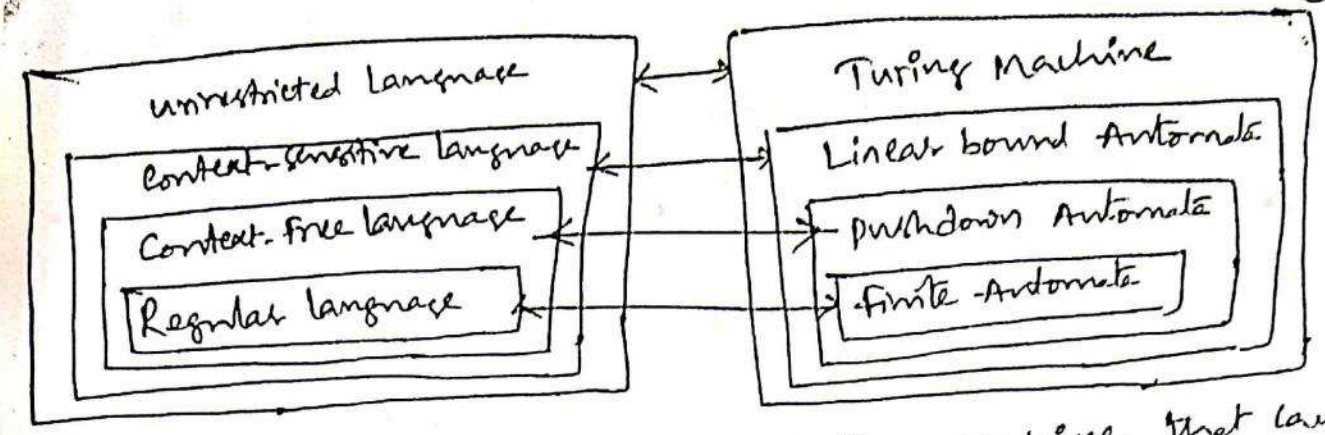


fig: The hierarchy of languages and the machine that can recognize the same as shown above fig.

. Every RG is context free, every CFG is context-sensitive and every CSG is unrestricted. So the family of regular language can be recognized by any machine.

. CFL's are recognized by pushdown automata, Linear bounded automata and Turing Machines.

. CSLs are recognized by Linear bounded automata and Turing machines

. Unrestricted Languages are recognized by only Turing machine

Push Down Automata (PDA)

(68)

PDA – FA + Stack (memory element)

PDA = $(Q, \Sigma, \delta, q_0, Z_0, F, \Gamma)$ (7 Tuple)

PDA → DPDA, NDPDA

Q = finite set of states

Σ = input symbol

δ = Transition function

q_0 = Initial state

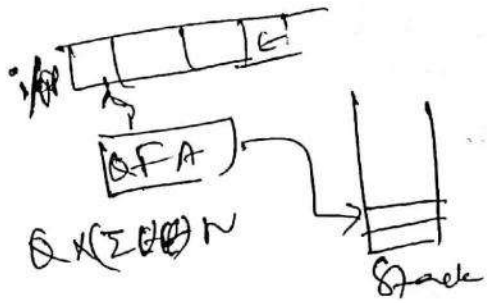
Z_0 = Bottom of the stack

F = set of final states

Γ = stack alphabet.

DPDA: $\delta : Q \times \{\Sigma \cup \epsilon\} \times \Gamma \to Q \times \Gamma$
(State, Input, Top)

NDPDA $\delta : Q \times \{\Sigma \cup \epsilon\} \times \Gamma \to 2^{(Q \times \Gamma)^*}$
(State, Input, Top most)



Ex: $a^n b^n \mid n \geq 1$.

(a, a/aa)
(a, Z0/aZ0)
(ε, Z0/ε)
(b, a/ε)

State transition Diagram

aabb ε

aabb ε

DPDA

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, \epsilon)$
$\delta(q_1, b, a) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, Z_0) = (q_f, Z_0)$ or (q_1, ϵ)

transition function

PDA – Final state / Empty stack.

(q_f, Z_0) – accept by final state.

(q_1, ϵ) – Acceptance by empty stack

$\{w | n_a(w) = n_b(w)\}$ [number of a's must be equal].

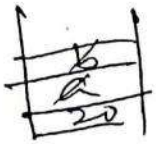
DPDA

Ex: a b bbaa
aabb baba
abab

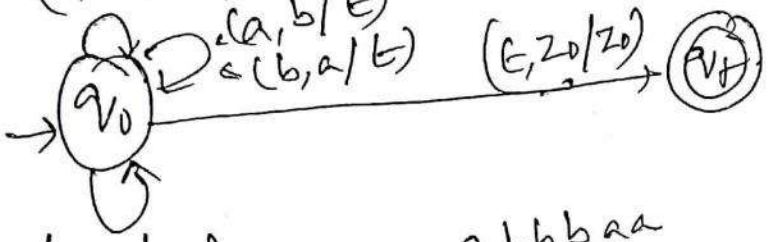
baba

abab

abba

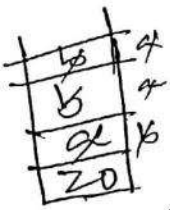


$(b, Z_0/bZ_0)$
$(a, Z_0/aZ_0)$



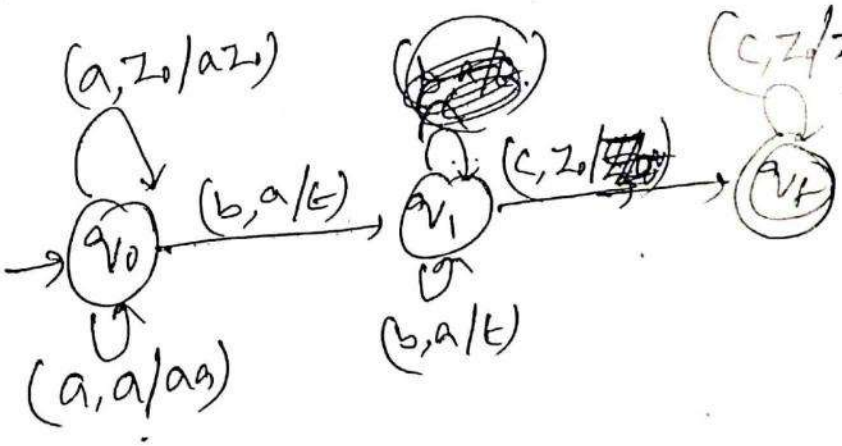
$(a, a/aa)$
$(b, b/bb)$

abbbaa



$a^n b^n c^m \mid n, m \geq 1$.

PDA
a^n → push
b^n → pop
c^m → in final state

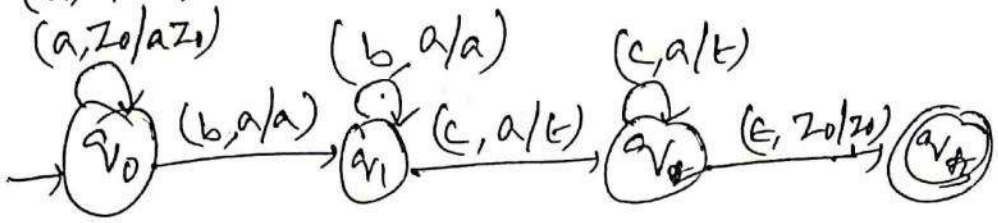


$a^n b^m c^n \mid n, m \geq 1$

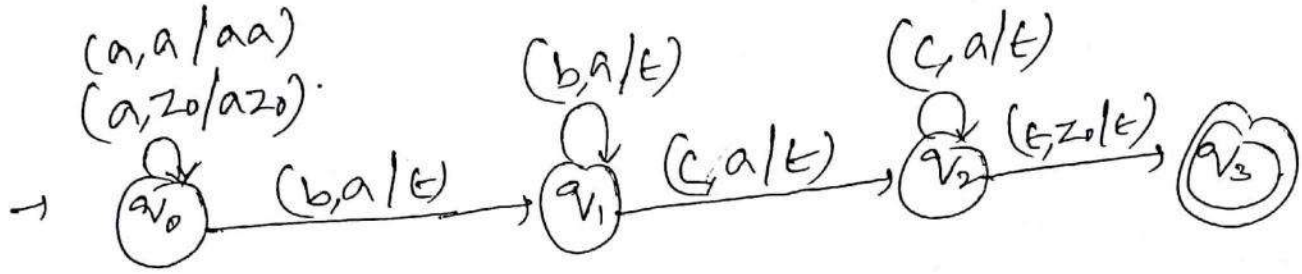
dont want count b

aaabbccc

Sol

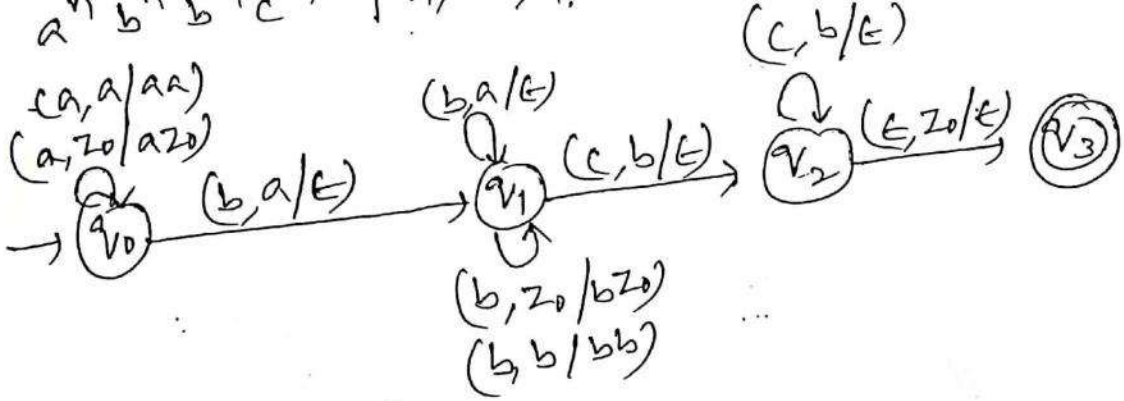


$a^{m+n} b^m c^n \mid m, n \geq 1$



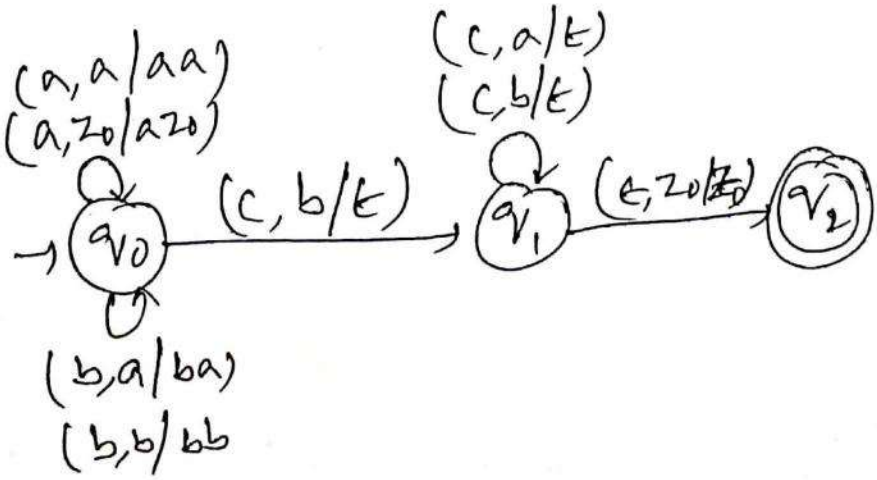
$a^n b^{m+n} c^m \mid n, m \geq 1$

Sol $a^n b^n b^m c^m \mid n, m \geq 1.$



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

Sol



- $a^n b^n c^m d^m \mid n, m \geq 1$
- $a^n b^m c^m d^n \mid n, m \geq 1$

- $a^n b^m c^n d^m \mid n, m \geq 1$ ✗ is not CFL not PDA

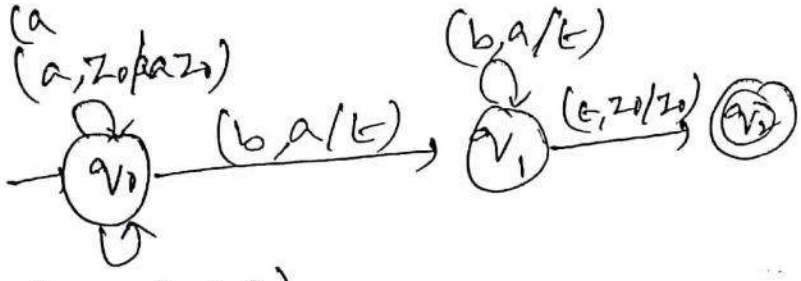
- $a^n 2^n \mid n \geq 1$

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

1) abb, aabbbb, aaabbbbbb, ...

two solutions: for every a – push two a's.
for every b – pop two b's

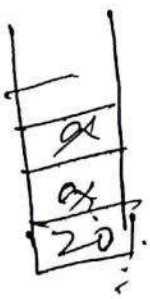
0



bb, bb,

for even b

aabbbb←

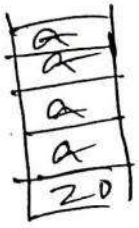


Ex:- $a^n b^n c^n \mid n \geq 1$ ✗ not a PDA

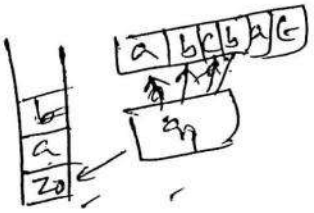
Sol:-

one possibility

✗ aa bb cc

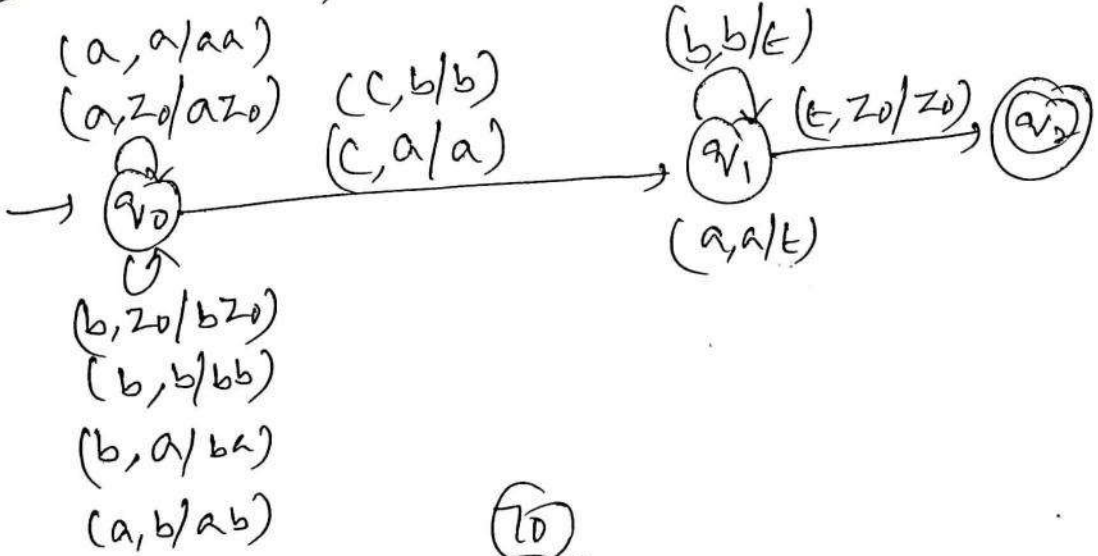


for every a → two alg push.



Ex:- ② $wcw^R \mid w \in (a,b)^+$

Sol:- abcba, abbcbba

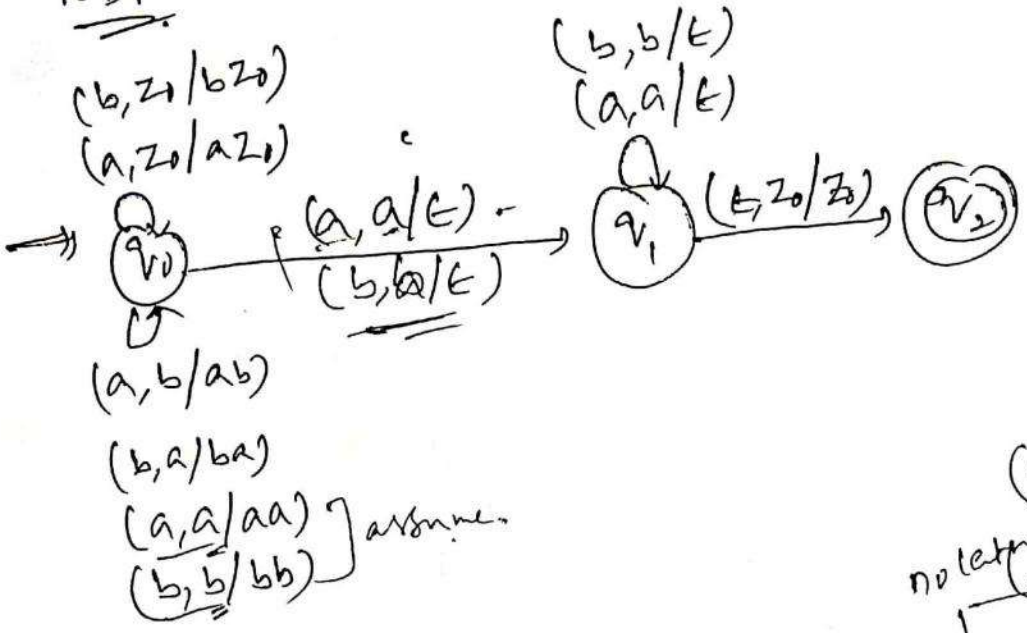


10

Ex:- $ww^R \mid w \in (a+b)^+$ → whenever

Sol:- $\underbrace{aba}_{w}\underbrace{aba}_{w^R}$

NDPDA $ww^R \mid w \in (a+b)^+$



abba

Instantaneous Descriptions

aaaa

$(q_0, aaaa, Z_0)$

(q_0, aaa, aZ_0)

no centre | centre

(q_0, aa, aaZ_0) (q_1, aa, Z_0) ×

no centre | centre

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

no centre | centre

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

(q_2, Z_0)

abbaab

aabbaa

w w

baaaab

abbabba

w w^R

Center

abba

w w^R

NDPDA

Center has come — pop

Center not come — push

Clearance → getting → centre

abba.

UNIT-VI

Language decidability

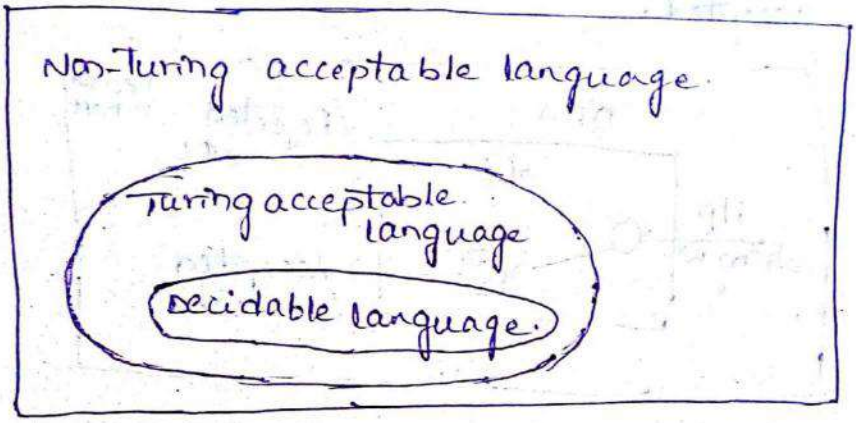
* Introduction
* Examples

Introduction:-

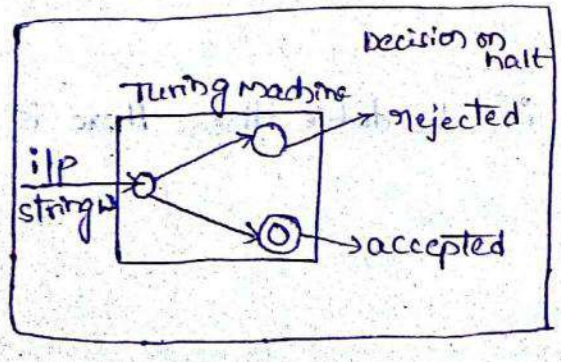
- Decidable problem:-

* A language is called Decidable (or) recursive if there is a turing machine which accepts and halts on every ilp string "w".

* Every decidable language is a turing acceptable.



* A decision problem 'p' is decidable if the language 'L' of all "yes" instances to 'p' is decidable.
* for a decidable language, for each ilp string, the turing machine halts either at the accept (or) the reject state



Examples:-

1) find out whether the following problem is decidable (or) not.

Is a number 'm' prime?

Sol:- Prime numbers = {2, 3, 5, 7, 11, 13, 17, 19, ----}

divide the number 'm' by all the numbers b/w

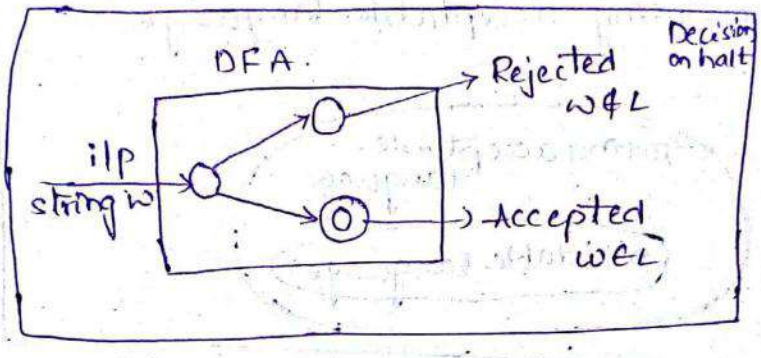
2 and m/2 · starting from 2.

If any of these numbers produce a remainder 0. then it goes to the rejected state. otherwise it goes to the accepted state. so, here the answer could be made by YES (or) NO.

Hence, it is a decidable problem.

2) Given a Regular language 'L' and string 'w', how can we check if $w \in L$.

sol:- Take the DFA that accepts 'L' and check if w is accepted.



Some more decision problems are.

i) Does DFA accept the empty language.

ii) Is $L_1 \cap L_2 = \phi$ for regular sets.

iii) If a language L is decidable then its complement L' is also decidable.

iv) If a language is decidable then there is a turing machine for it.

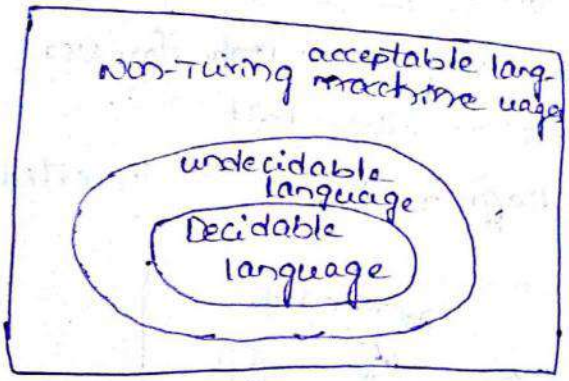
Undecidable problems:-

Introduction:-

* for an undecidable language there is no "TM" which accepts the language and makes a decision for every i/p string 'w'.

* A decision problem 'p' is called undecidable if the language 'L' of all 'YES' instances to 'p' is not decidable.

undecidable languages are not recursive languages but, sometimes they may be recursive enumerable languages.



Examples:—

i) The halting problem of turing machine.

ii) The mortality problem.

iii) The mortal matrix problem.

iv) The post correspondence problem [pcp]

i) The halting problem:

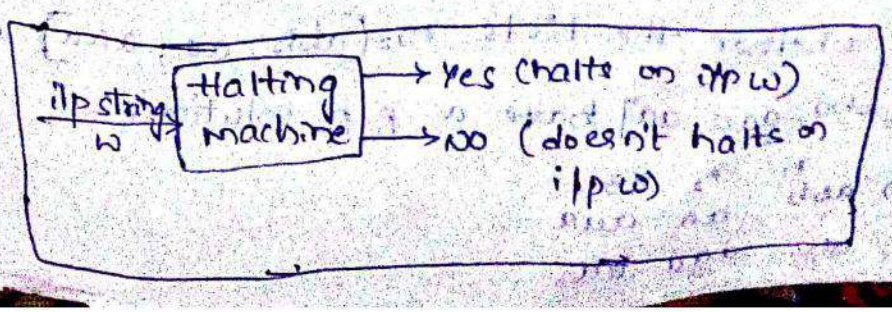
The halting problem i/p: a turing machine and the i/p string w.

problem: Does the turing machine finish computing of the string 'w' in a finite no. of steps? The answer must be either yes (or) NO.

Proof:— At first, we will assume that a turing machine exists to solve, the problem. we will show and then it is contradicting it self.

we will call this turing machine as a halting machine that produces a yes (or) NO. in a finite amount of time.

If the halting machine finishes in a finite amount of time then the o/p comes as YES. otherwise, as NO

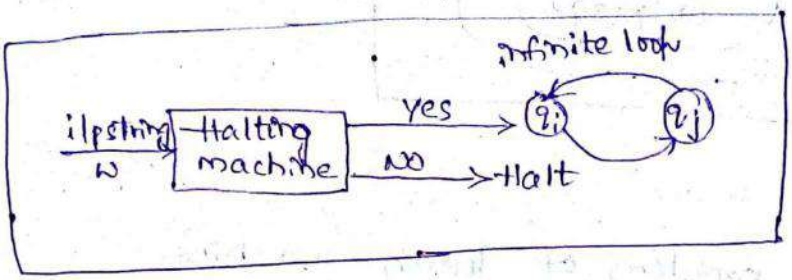


Now, we will design an inverted halting machine as.

i) If HM returns YES then loop forever.

ii) If HM returns NO then halt.

The following block diagram shows the inverted halting machine.



further, a machine "HM" which ilp itself is constructed as follows.

i) If HM halts on ilp loop forever.

ii) Else Halt

∴ Here, we have got a contradiction. Hence, the halting problem is undecidable.

**Post correspondence problem (pcp):—

- It was introduced by "Emil post" in 1946 is as undecidable decision problem.

The pcp problem over an ilp alphabet 'Σ' is stated as follows.

Given the following two lists, M, and N. of non-empty strings over Σ

$$M=(x_1, x_2, x_3 \cdots x_n)$$

$$N=(y_1, y_2, y_3 \cdots y_n)$$

We can say that there is a pcp solution if for some $i_1, i_2, i_3 \cdots i_K$ where $1 \le i_K \le n$, The condition.

$$x_{i_1} x_{i_2} x_{i_3} \cdots x_{iK} = y_{i_1} y_{i_2} y_{i_3} \cdots y_{iK}$$ satisfies

Ex:- Find whether the lists $M=[abb, aa, aaa]$ and $N=[bba, aaa, aa]$ have a pcp solution.

Sol:-

	x_1	x_2	x_3
M	abb	aa	aaa
N	bba	aaa	aa

Here $x_2 x_1 x_3 = aaabbaaa$

$y_2 y_1 y_3 = aaabbaaa$

we can see that $x_2 x_1 x_3 = y_2 y_1 y_3$.

Hence, the solution is $i=2, j=1, K=3$.

2) find whether the list $M = [ab, bab, bbaaa]$ and $N = [a, ba, bab]$ have a pcp solution.

sol:-

	x_1	x_2	x_3
M	ab	bab	bbaaa
N	a	ba	bab

In this case, there is no solution. because

$|x_2 x_1 x_3| \neq |y_2 y_1 y_3|$ length's are not same.

Hence, it can be said that this pcp is a undecidable problem.

Modified post correspondence problem:-

Given two lists $M = x_1, x_2, x_3 \cdots x_n$ and $N = y_1, y_2, y_3 \cdots y_n$

Given a set of pairs of strings $\{(x_1, y_1), (x_2, y_2), \cdots (x_n, y_n)\}$

then the solution is an instance such that,

$$x_1 x_{i_1} x_{i_2} \cdots x_{i_n} = y_1 y_{i_1} y_{i_2} \cdots y_{i_n}$$

that means the pair (x_1, y_1) is forced to be at the begining of the strings.

Ex:-

	x_1	x_2	x_3
M	11	100	111
N	111	001	11

sol:- ∴Then the solution is $x_1 x_2 x_3 = y_1 y_2 y_3$.

$x_1 x_2 x_3 = 11100111$

$y_1 y_2 y_3 = 11100111$

That means it is essential to have x_1 & y_1 at the beginning of list.

P and Np Classes :-

* P-problems
* Np-problems
* P vs Np

- **P-problems:-**

* P is the class of problems that can be solved by deterministic algorithm in a polynomial type time $p(n^k)$ where 'n' is the size of i/p string.
* P-problem consist of a language accepted by deterministic Turing machine that runs in polynomial amount of time.

Ex:- 1) shortest path problem
2) Equivalence of NFA and DFA.
3) shortest cycle in a graph.
4) sorting algorithms

- **Np-problems:-**

* Np-problem is a class of problems that can be solved by Non-deterministic algorithms in a polynomial time $p(n^k)$ where 'n' is the size of i/p string.
* Np-problems consists of a language accepted by Non-deterministic Turing machine that runs in a polynomial amount of time.

Ex:- 1) Travelling salesman problem.
2) subgraph isomorphism

Np-problem classified into two types:

i) Np-hard problem.
ii) Np- complete problem.

- **Np-hard problem:-**

If there is a language x such that every Language y in Np can be polynomially reduceable to x. and we cannot prove that x is in Np. then x is said to be Np-hard problem.

Ex: Turing machine halting problem.

- **Np- Complete problem:-**

If there is a Language x such that every Language

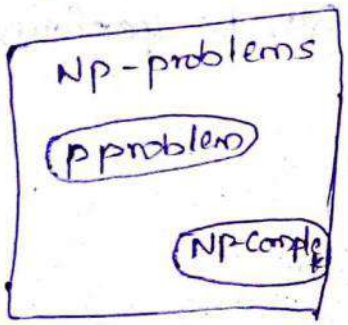
y in NP can be polynomially reducible to x and we can prove that x is in NP then x is said to be NP-complete problems.

Eg:- 1) Travelling salesman problem.
2) Subgraph isomorphism.

P VS NP :-

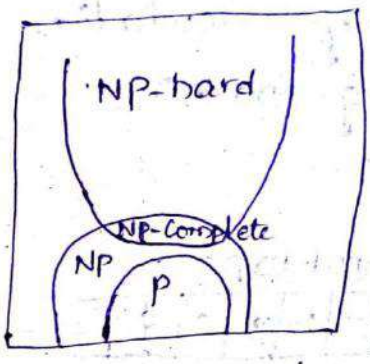
1. Ladner's theorem:

(a) $P \neq NP$



2. Euler's theorem :-

(a) $P \neq NP$



(b) $P = NP$