MODULE -3
PUSHDOWN AUTOMATA

A Grammar is nothing but a set of rules to define valid sentences in a languages.Context free grammar generates contextfree languages.Contextfree languages have great practical significance in defining programming languages and in simplifying the translation for programming languages.Context free grammars are useful for describing arithmetic expressions with arbitrary nesting of balanced paranthesis.Neither of these aspects of programming languages can be represented by regular expression.

A Context free grammar is defined as follows: G=(V,T,S,P) where

V->set of nonterminals

T->set of terminals

S->Start symbol

P->set of rules or productions in P

G is context free and all productions in P have the form α->β where α ε V and β ε (V UT)*.A language 'L' is said to be context free if and only if there is a context free grammar G such that L=L(G).

Leftmost and Rightmost derivation:

A derivation is said to be leftmost if in each step the leftmost variable in the sentential form is replaced. If we replace the rightmost variable then it is rightmost derivation.

Derivation Tree:

It is n ordered tree in which nodes are labeled with left side of the production and the child represent its corresponding right side of the production.

Let G=(V,T,S,P) be a CFG. An ordered tree is a derivation tree for G if and only if it has following properties:

1.The root is labeled S

2.Every leaf has a label from T U{ ε}

3.Every non-leaf nodes has a label from V.

4.If a vetex has label A ε V and its chidren nodes are labeled $a_1,a_2 \ldots a_n$, then P must contains a production of the form A-> $a_1,a_2 \ldots a_n$

A tree is said to be partial derivation tree if its second property is replaced by

1.Every leaf has a label from VUTU{ ε }.

Yield of a tree:

The strings of symbols obtained by reading the leaves of the tree from left to right omitting ε's is said to be yield of a tree.

A Context free grammar G is said to be ambiguous if it generates two parse trees for same sentence.

Reduction of Contextfree Grammar:

There are several ways in which one can restrict the format of CFG without reducing the language generation power of CFG.

- **Eliminate useless symbols**
- **Eliminate unit productions**
- **Eliminate ε-productions**

a. **Eliminate Useless symbols:**

Identify those symbols which don't play any role in the derivation of any string 'w' in L and then eliminate the identified productions which contains useful symbols from the CFG.

- Identify non-generating symbols in given CFG and eliminate those productions whih contains non-generating symbols
- Identify non-reachable symbols and eliminate those productions which contains non-reachable symbols**.**

b. **Eliminate unit productions**

Unit productions is a productions of the form NT->NT

Algorithm:

Removal of unit productions ->

While(there exists a unit productions, A->B)

{

SELECT A UNIT PRODUCTIONS A->B , such that there exists a production, B-> α, where α is a terminal.

For (every non-unit productions,B-> α)

Add productions A-> α to the grammar.Eliminate A->B from the grammar.

}

c)Removal of ε-productions and Nullable Nonterminal

A nonterminal N is nullable if

1.There is a production N-> ε

2.There is a derivation that starts at N and leads to ε .

To eliminate ε-productions from a grammar G we use the following technique:

If A is a production to be eliminated then we look for all the productions whose RHS contains A and replace each occurances of A by ε in each of these productions to obtain the non- ε productions.Now these non- ε productions must be added to the grammar.

Normal forms of CFG:

There are two normal forms:

- **Chomsky Normal Form(CNF)**
- **Greibach Normal Form(GNF)**

1.Chomsky Normal Form(CNF)

A CFG is CNF if all the productions are of the form

NT->string of exactly two NT's or of the form NT->a terminal.

2.Greibach Normal Form (GNF)

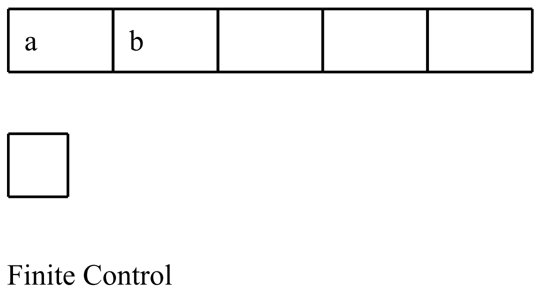
A CFG is said to be in GNF if all the productions have the form A->ax where a ε T and x ε v*.

PUSHDOWN AUTOMATA

FA is not capable to recognize the context free languages such as $\{wcw^R / w\ \varepsilon \sum^* \}$ bcoz by reading a string in this language from left to right, automata must remember the string before it encounter symbol ‘c’ and then compare it to string after ‘c’.But FA is not capable to remember anything.So we extend FA by adding an auxiliary storage to accept context free languages.

PDA is essentially a FA with control of both input tape and stack to store what it has read.PDA has 3 things:

- An input tape
- A finite control
- A stack



PDA has an input tape from which finite control reads the input and at the same time it can read symbol from the stacktop. Finite control determines what is the next state and what will happen with the stack top. Each move of the control unit is determined by the current input symbol as well as the symbol currently on the top of the stack.The result of the move is a new state of the control unit and a change in the top of the stack.

A PDA is mathematically defined as $M=(Q,\sum ,\zeta ,\delta, q0, Z,F)$ where

Q ->finite set of internal states of the control unit

$\sum$ ->alphabet set

Z -> finite set of symbols called stack alphabets

Δ -> a transition function which maps $Q^* (\sum \cup \{ \varepsilon\}) * \zeta$ -> finite subset of $Q * \zeta$

$Z\ \varepsilon\ \zeta$ -> stack starting symbol

F -> set of final states.

Each move of the control unit is determined by the current input symbol as well as by the symbol currently on the top of the stack. The result of the move is a new state of the control unit and a change in the top of the stack.Thus the relevant factors at any time are the current state of control unit, the unread part of the input string and current contents in the stack.The triplet (q,w,u) is called an instantaneous description of a PDA.A move from one ID to another will be denoted by ‘l

Language accepted by M is the set of all the strings that can put ‘M’ into a final state at the end of the string.

Set of all inputs for which some sequences of moves causes PDA to empty its stack. This language is referred to as **language accepted by empty stack**.

The second way of defining the language accepted is similar to the way a FA accepts inputs.That is designate some states as final states and define the accepted language as the set of all inputs for which some choices of moves causes the PDA to enter a final state. This language is referred to as **language accepted by final state.**

1.Construct a PDA to accept $a^n b^n$;n>=1 by final accepting state

Let M be a PDA .q_0 be the initial state.If ‘a’ is encountered at state q_0, then push it into the stack.

δ (q0 ,a ,Z_0) =(q0, aZ_0)

δ (q0 ,a ,aZ_0) =(q0, aZ_0)

If a symbol ‘b’ is encountered then change state to q_1 and pop one ‘a’ from the stack.

δ (q0 ,b,aZ_0) =(q1, Z_0)

Process b’s

δ (q1 ,b, a) =(q1, Z_0)

Final state

δ (q1 , ε, Z_0) =(q_2, Z_0)

PDA equivalent to a CFG

A PDA can simulate a derivation in the grammar.PDA starts by pushing the start symbol ‘S’ on the stack.Two types of moves are made by the PDA after ‘S’ is placed on the stack as follows:

1.Pop a terminal symbol from the stack if it matches the next input symbol.Both the symbols are then discarded.

2.Replace a variable A on the top of the stack by the right side of the production ,

A-> α

Repeat the steps 1 and 2 untill end of the input string is reached.Then if the PDA is in final state, it will accept the input.

Construct a PDA that accepts the language given by the grammar G, S->aSbb/a

Let M be the PDA With Q={ q0,q1,q2} and F={q2}

State	Input	Stack symbol	Moves
q0	ε	Z_0	(q1 ,S $Z_{0)}$
q1	ε	S Z_0	(q1,a) (q1,aSbb)
q1	a	a	(q1, ε)

q1	b	b	(q1, ε)
q1	ε	Z_0	(q2, Z_0)