# Context Free Language

# Context Free Grammar:

- A context-free grammar can be described by a four-element tuple $(V,\Sigma,R,S)$ where

  - *V (N)* is a finite set of variables (which are non-terminal); represented by capital letters.

  - $\Sigma$ (T) is a finite set ( (disjoint from V) of terminal symbols; represented by small letters or special symbols like $, +,*,0,1,(,), etc.

  - *R (P)* is a set of production rules where each production rule maps a variable to a string $s \in (V \cup \Sigma)^*$

  - *S* (which is in V) which is a start symbol

- Eg:

  $S \rightarrow e$

  $S \rightarrow 0S$

  $S \rightarrow 1S$

String: 001

# Leftmost Derivation & Rightmost Derivation

**Leftmost Derivation:**

- The process of deriving a string by expanding the leftmost non-terminal at each step is called as **leftmost derivation**.

The geometrical representation of leftmost derivation is called as a **leftmost derivation tree**.

**Rightmost Derivation:**

- The process of deriving a string by expanding the rightmost non-terminal at each step is called as **rightmost derivation**.
- The geometrical representation of rightmost derivation is called as a **rightmost derivation tree**.

- Eg: consider grammar:

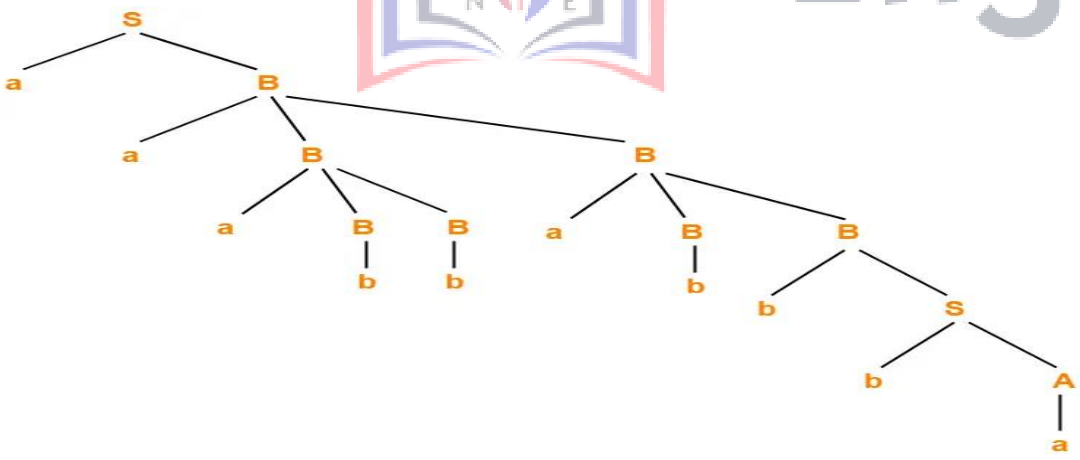
$S \rightarrow aB / bA$

$S \rightarrow aS / bAA / a$

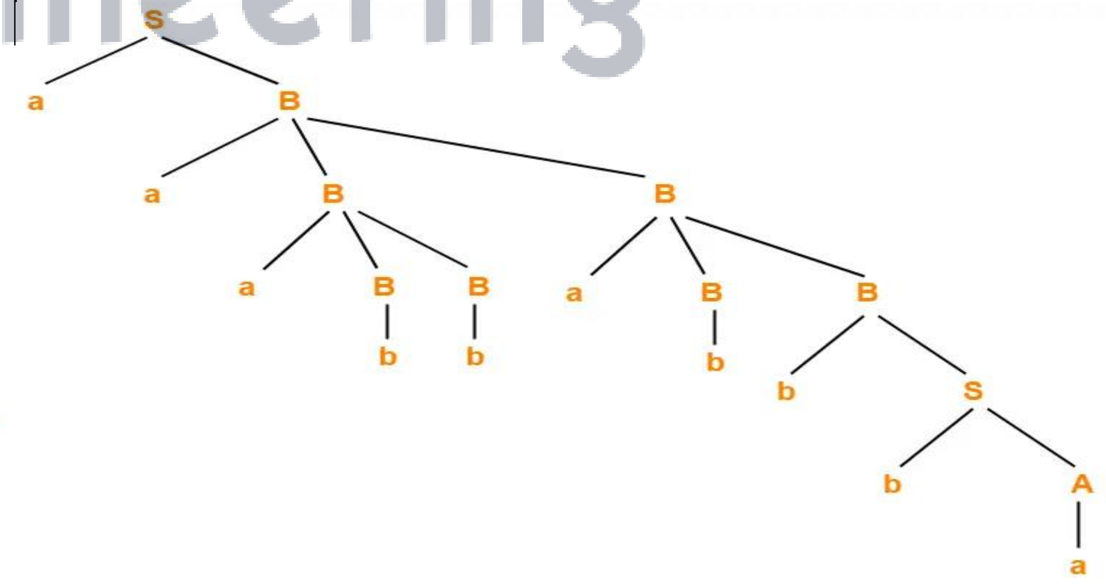
$B \rightarrow bS / aBB / b$

Input string w = aaabbabbba

| Leftmost Derivation | | Rightmost Derivation | |
|---|---|---|---|
| S → a**B** | | S → a**B** | |
| → aa**B**B | (Using B → aBB) | → aaB**B** | (Using B → aBB) |
| → aaa**B**BB | (Using B → aBB) | → aaBaB**B** | (Using B → aBB) |
| → aaab**B**B | (Using B → b) | → aaBaBb**S** | (Using B → bS) |
| → aaabb**B** | (Using B → b) | → aaBaBbb**A** | (Using S → bA) |
| → aaabba**B**B | (Using B → aBB) | → aaBa**B**bba | (Using A → a) |
| → aaabbab**B** | (Using B → b) | → aa**B**abbba | (Using B → b) |
| → aaabbabb**S** | (Using B → bS) | → aaa**B**Babbba | (Using B → aBB) |
| → aaabbabbb**A** | (Using S → bA) | → aaa**B**babbba | (Using B → b) |
| → aaabbabbba | (Using A → a) | → aaabbabbba | (Using B → b) |



Leftmost Derivation Tree



Rightmost Derivation Tree

# Ambiguous Grammar:

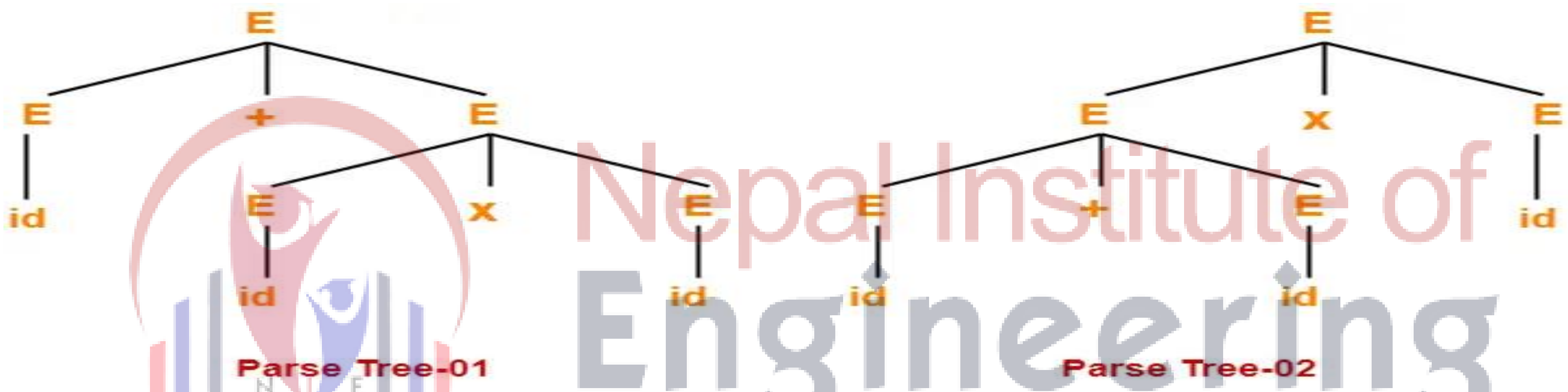
- A grammar is said to be ambiguous if for any string generated by it, it produces more than one-
  - Parse tree
  - Or derivation tree
  - Or syntax tree
  - Or leftmost derivation
  - Or rightmost derivation
- Consider the following grammar-

$$E \rightarrow E + E \,/\, E \times E \,/\, id$$

Let us consider a string w generated by the grammar-

- w = id + id x id

Now, let us draw the parse trees for this string w.






Since two parse trees exist for string w, therefore the grammar is ambiguous.

# Chomsky Normal Form(CNF):

CNF form:

A-> BC {V->VV}

A-> a {V->T}

- Simplifications:

1. **Elimination of epsilon productions:** A->e; if grammar has n nullable symbols then there is 2n possible combinations.

2. Elimination of Unit productions: A->B

3. Elimination of useless symbols: non-generating symbols and non-reachable symbols. All terminals are generating symbols. If RHS has generating symbols then LHS also be generating symbols.

Eg: S->AB|a

A->b then generating symbols={a,b,S,A} and non generating symbols={B}

After removing non generating symbols the grammar is:

S->a

A->b

**Non reachable :** starts symbols of grammar is reachable symbol. LHS of production is reachable then RHS of the production is also reachable symbols.

Eg: S->a

A-> b then reachable symbols {S,a} and non reachable =(A,b}. After removing non reachable from the grammar is

S->a;

Eg: S->aA |aBB

A->aAA|e

B->bB|bbC

C->B

Solution:

i. Ellimination of e-productions:

A->e after removing nullable symbols A is

S->aA|aBB|a

A->aAA|aA|a

B->bB|bbC

C->B

ii. Elimination of unit productions:

C->B, after removing unit production:

S->aA|aBB|a

A->aAA|aA|a

B->bB|bbC

C->bB|bbC

iii. Elimination of usless symbols:

generating symbols={a,b,S,A}

non generating symbols={B,C}

After removing non generting symbols

S->aA|a

A->aAA|aA|a

Reachable symbols={S,a,A}

iv. New variable for terminal

P->a

S->PA|a

A->PAA|PA|a

v. CNF is:

P->a

S->PA|a

A->PX|PA|a

X->AA

# Greibach Normal Form GNF:

Form:

A->aBC (V->T.any no. of V)

A->b (V->T)

- Eliminate left recursion in gnf.
- the given grammar is in CNF.

# Pumping Lemma for CFG:

- To prove certain languages are not context-free.
- Let L be a CFL and n be a constant.
- Any string z in L, $|z| \geq n$.
- Split z= uvwxy such that:
  - $|vwx| \leq n$
  - $Vx \neq e$ or $|vx| \geq 1$
  - For all $i \geq 0$, $uv^iwx^iy \in L$

Show that L= $a^nb^nc^n | n>=1$ is not CFL

Let $z = a^nb^nc^n$, $|z| \geq n$

Split z =uvwxyz

$u=a^n$

$vwx=b^n$, $|vwx| \leq n$

$vx= b^{n-m}$, $|vx| \geq 1$, $m<n$

$y=c^n$

$uv^iwx^iy= \quad uvv^{i-1}wxx^{i-1}y$

$=uvwx(vx)^{i-1}y$

$=a^nb^n(b^{n-m})^{i-1}c^n$

$=a^nb^{n+ ni-n-mi+m}c^n$

$=a^nb^{ni-mi+m}c^n$

Pick i=0, then

$Uv^iwx^iy=a^nb^mc^n \notin L$ hence the given language is not CFL.

# Push Down Automata (PDA)

- Pushdown Automata is a finite automata with extra memory called stack which helps Pushdown automata to recognize Context Free Languages.
- Mathematically a Pushdown Automata (PDA) is defined with 7 tuple , M=(Q, ∑, Γ, q0,Z,F, δ) where,

Q is the set of states

∑is the set of input symbols

Γ is the set of pushdown symbols (which can be pushed and popped from stack)

q0 is the initial state

Z is the initial pushdown symbol (which is initially present in stack)

F is the set of final states

δ is a transition function which maps Q x {Σ ∪ ∈} x Γ into Q x Γ*. In a given state, PDA will read input symbol and stack symbol (top of the stack) and move to a new state and change the symbol of stack.

# Instantaneous Description (ID):

- A ID is a triple $(q, w, \alpha)$, where:
  1. q is the current state.
  2. w is the remaining input.
  3. $\alpha$ is the stack contents, top at the left.
- **Turnstile notation**

  $\vdash$ sign is called a "turnstile notation" and represents one move.

  $\vdash^*$ sign represents a sequence of moves.

  Eg- $(p, b, T) \vdash (q, w, \alpha)$

# Language of PDA:

1. Acceptance by final state:

Let P = $(Q, \sum, \Gamma, q0, Z, F, \delta)$ be a PDA then

L(P)= $\{w \mid (q0,w,z0) \vdash^* (qF,e, j)$; where qF= final state and j= any symbols in stack.

2. Acceptance by empty stack:

Let P = (=$(Q, \sum, \Gamma, q0, Z, F, \delta)$ be a PDA then

L(P)= $\{w \mid (q0,w,z0) \vdash^* (qF,e, e)$;

# Problem: construct a PDA for L={$0^n1^n$|n≥1

Solution:

L ={01, 0011, 000111,00001111,……. } equal no. of 0 followed by equal no of 1.

Let us consider an input = 000111e

Transition:

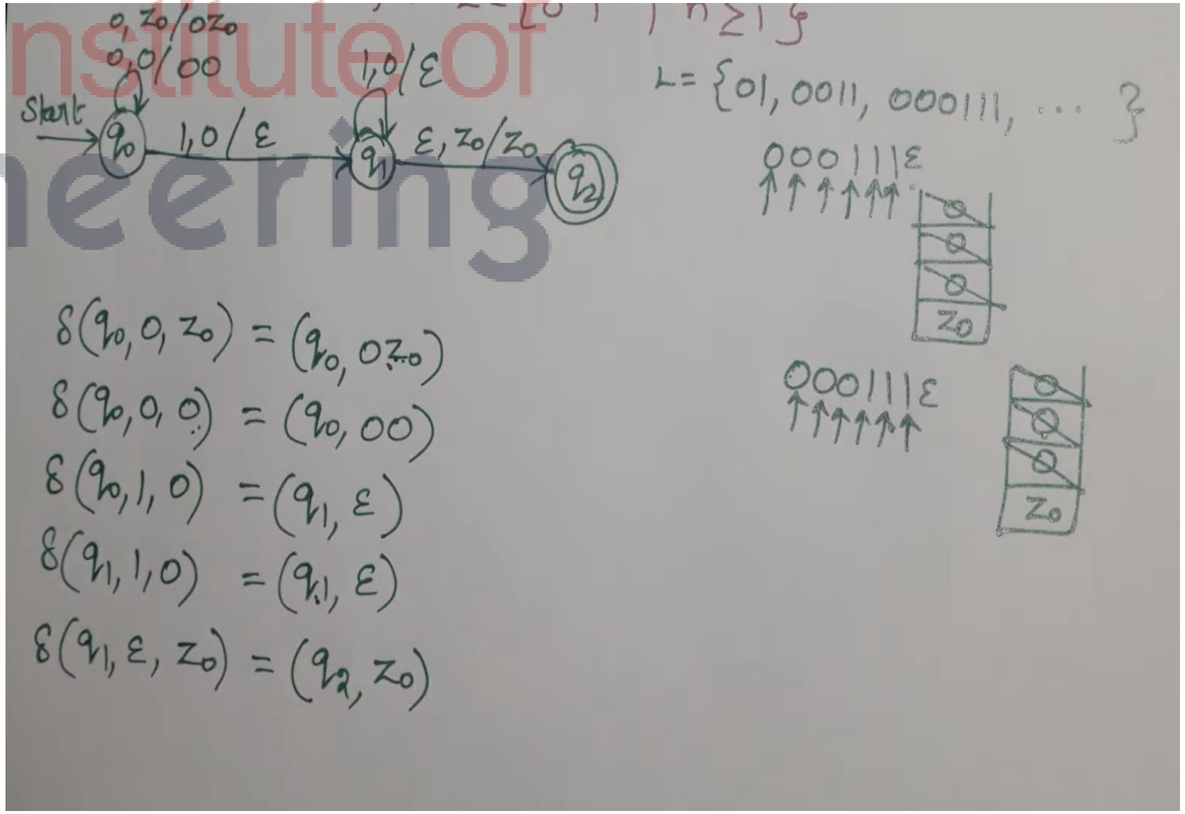
δ(q0,0,z0)=(q0,0z0) // push into stack

δ(q0,0,0)=(q0,00) //push into stack

δ(q0,1,0)=(q1,e) // pop from stack

δ(q1,1,0)=(q1,e) // pop from stack

δ(q1,e,z0)=(q2,z0)



# Equivalence of CFG and PDA

- CFG and PDA are equivalent in power.
- a CFG generates a context-free language and a PDA recognizes a context-free language.
- A language is context-free iff some pushdown automaton recognizes it.
- **L(G) = L(P)**

# Algorithm to find PDA corresponding to a given CFG

- **Input** – A CFG, G = (V, T, P, S)
- **Output** – Equivalent PDA, P = (Q, ∑, S, δ, $q_0$, I, F)
- **Step 1** – Convert the productions of the CFG into GNF.
- **Step 2** – The PDA will have only one state {q}.
- **Step 3** – The start symbol of CFG will be the start symbol in the PDA.
- **Step 4** – All non-terminals of the CFG will be the stack symbols of the PDA and all the terminals of the CFG will be the input symbols of the PDA.
- **Step 5** – For each production in the form **A → aX** where a is terminal and **A, X** are combination of terminal and non-terminals, make a transition **δ (q, a, A)**.

# Problem

- Construct a PDA from the following CFG.

**G = ({S, X}, {a, b}, P, S)**

- where the productions are –
- **S → XS | ε , X → aXb | Ab | ab**

Solution:

- Let the equivalent PDA,
- P = ({q}, {a, b}, {a, b, X, S}, δ, q, S)
- where δ –
- δ(q, ε , S) = {(q, XS), (q, ε )}
- δ(q, ε , X) = {(q, aXb), (q, Xb), (q, ab)}
- δ(q, a, a) = {(q, ε )}
- δ(q, b, b) = {(q, ε )}

# Properties of Context Free Languages

- **Union :** If L1 and L2 are two context free languages, their union L1 ∪ L2 will also be context free.
- **Concatenation :** If L1 and If L2 are two context free languages, their concatenation L1.L2 will also be context free.
- **Kleene Closure :** If L1 is context free, its Kleene closure L1* will also be context free.
- **Intersection and complementation :** If L1 and If L2 are two context free languages, their intersection L1 ∩ L2 need not be context free