/ backspace form

VIJAYANT PAGE NO. DATE

UNIT 3- Context Free grammar

Grammar (Generation)

↓

Language

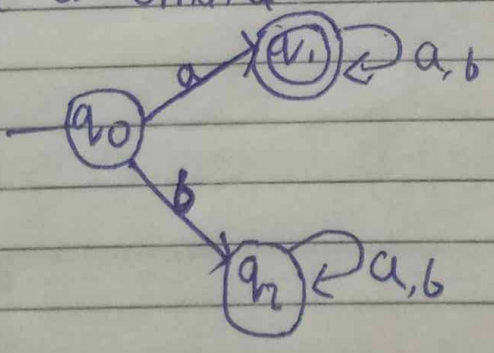
↓

Automata (Acceptor)

$L=\{a, aa, ab, aba, aabaa, abb\}$ → This language is infinite

Because we have infinite combination of string a & b.

Digram of finite automata of infinite state automata



This is a machine of Finite representation of infinite language

$L(M)=\{ w \mid w \in \Sigma^*, \delta^*(q_0 w) \in q_F \}$

for every w

'L' of 'M' is equal to such that 'w' belongs to transition closure.

only capital letters can be rewritten

VIJAYANT PAGE NO: DATE:

$S \rightarrow aA$

S can be rewrite as 'a' & 'A'

'A' can be rewrite as $(A \rightarrow aA / bA / \epsilon)$

Non terminal = capital symbols

String is not terminated

Terminal = Small case symbols / letter's

String is Terminated

Small case symbols are terminal, it is nothing but sigma or a, b, c...

★ Production Rule (we can Rewrite any symbol)

$\alpha \rightarrow \beta$

non Terminal

It can be sential form or tree Form.

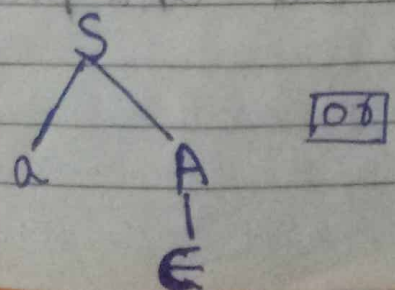
Derivation means From start symbol when we try to reach string symbol's or string generation.

Ex

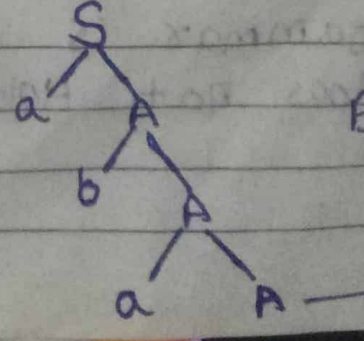
$S \rightarrow aA$

$A \rightarrow aA / bA / \epsilon$

Tree form for S



or



ϵ after we use it ends is used to terminate

$L(G) = \{w / w \in \Sigma^*, S \xrightarrow{*} w\}$

This means → Taking any no. of production from start

Grammar is defined by quadruple

$G = \{\Sigma, V_n, P, S\}$ 4 Tuple

where,

Σ → known as finite set of terminals / small case letter's, Ex a, b

V_n → Finit non-empty set of non-terminals / upper case ex. S, A

P → production Rule
Finit non-empty set of prodoction Rule

S → S stands for start Symbol.

Note:→

Ex $\alpha \rightarrow B$

α ↓ non terminal

If we not have the alfa then the Grammar is no invalid and production Rule does not Holds.

a → b

↓

Tends to

can be re-write as

Ex to solve → w = bbaababa

Consider the grammar

S → bB / aA

A → b / bS / aAA

B → a / aS / bBB

For the string w = bbaababa

S =

Left most Derivation.

S → bB

→ bbBB (using B → bBB)

→ bbaSB (using B → aS)

→ bbaaAB (using S → aA)

→ bbaabB (using A → b)

→ bbaabaS (using B → aS)

→ bbaababB (using S → bB)

→ bbaababa (using B → a)

Right most derivation

S → bB

→ bbBB (using B → bBB)

→ bbBa (using B → a)

→ bbaSa (using B → aS)

→ bbaaAa (using S → aA)

→ bbaabSa (using A → bS)

→ bbaabaAa (using S → aA)

→ bbaababa (using A → B)

problem - 2 :

consider the grammar

$S \rightarrow A1B$
$A \rightarrow 0A \mid \epsilon$
$B \rightarrow 0B \mid 1B \mid \epsilon$

For the string $W = 00101$, find
Left most derivation
Right most derivation
Parse Tree

Left most derivation.

$S \rightarrow A1B$
$\rightarrow 0A1B$ (using $A \rightarrow 0A$)
$\rightarrow 00A1B$ (using $A \rightarrow 0A$)
$\rightarrow 00\epsilon1B$ (using $A \rightarrow \epsilon$)
$\rightarrow 00\epsilon10B$ (using $B \rightarrow 0B$)
$\rightarrow 00\epsilon101B$ (using $B \rightarrow 1B$)
$\rightarrow 00\epsilon101\epsilon$ (using $B \rightarrow \epsilon$)
$\rightarrow 00101$

Right most derivation

$S \rightarrow A1B$
$\rightarrow A10B$ (using $B \rightarrow 0B$)
$A101B$ (using $B \rightarrow 1B$)
$A101\epsilon$ (using $B \rightarrow \epsilon$)
$0A101\epsilon$ (using $A \rightarrow 0A$)
$00A101\epsilon$ (using $A \rightarrow 0A$)
$00\epsilon101\epsilon$ (using $A \rightarrow \epsilon$)
00101

CFL = Context Free Language

T-0 → Unrestricted Grammar → very few rules

T-1 → context.

Formal grammer

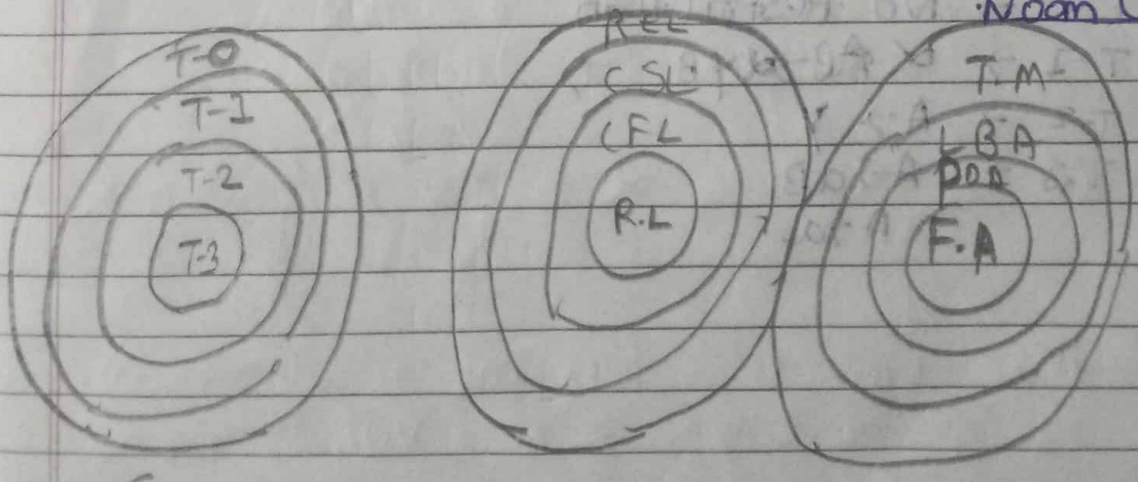
- generative grammer
- Analatical grammer

* Context Free grammer

In Formal language theory, A context free language is a language generated by Some context free grammer.

When the set of all CFL is identical to The set of Languages accepted by pushdown automata

Chomsky Heraschy of languages by Noam Chomsky in 195



Grammar Language Matchin

T-0 → Unrestricted Grammer
Very Few Restriction.

T-1 → Context Sensitive grammer → some Restriction
T-2 → Context Free gramer
T-3 → Regular grammer

REL - Recursive enumerable Language
CSL - Context ~~Free~~ sensitive language
CFL - context Free Language
RL - Regular Language.

T.M - Turing machine
LBA - Least Bounded Automata.
PDA - Pushdown Automata
F.A - Finite Automata

Production Rules

T-0 → No Restriction
T-1 → $\alpha A B \rightarrow \alpha \gamma B$
T-2 → $A \rightarrow \gamma$
T-3 → $A \rightarrow aB$
$A \rightarrow a$.

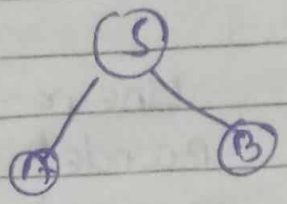
Grammar	Language	Automaton	Machine
Type-0	Recursively Enumerable Unrestected grammar	Turing machine	No Restriction
Type-1	context - sensitive	Linear - Bounded non deterministic Turing machine	$\alpha A \beta \rightarrow \alpha \gamma \beta$ Grammer with some Restrictions
Type-2	Context Free	Non-Deterministic pushdown Automata	$A \rightarrow \gamma$ Grammer
Type-3	Regular	Finite set-Automata	$A \rightarrow aB$ $A \rightarrow a$

CFG → Context Free Grammar.

* = closure

sentential Form

$S \to AB \;/\; A \to Baa \;/\; A \to \epsilon$



when you have S as Root Node

If a partial derivation tree contains the root 'S' it is called sentential form and above subtree is a sentential tree.

$G = (V, T, P, S)$ be a CFG
then any string w in $(V \cup T)^* / S \overset{*}{\Rightarrow} w$ is a Sentential form

If $S \overset{*}{\Rightarrow} \alpha$ where $\alpha \in (V \cup T)^*$, then α is a sentential form
V= is set of Variable
T= is set of Terminals

If $S \overset{*}{\underset{lm}{\Rightarrow}} \alpha$ where $\alpha \in (V \cup T)^*$, then α is a left sentential form.

Hear Lm is left most Derivation
If $S \overset{*}{\underset{rm}{\Rightarrow}} \alpha$ where $\alpha \in (V \cup T)^*$, then α is a right sentential form

Derivation in sentential form

ex Grammar Given

$S \rightarrow A1B$
$A \rightarrow 0A \mid \epsilon$
$B \rightarrow 0B \mid 1B \mid \epsilon$

String 1001

L.M

① $S \rightarrow A1B$
② $\rightarrow \epsilon 1B$ (using $A \rightarrow \epsilon$)
③ $\rightarrow \epsilon 10B$ (using $B = 0B$)
④ $\rightarrow \epsilon 100B$ (using $B = 0B$)
⑤ $\rightarrow \epsilon 1001B$ (using $B = \epsilon$) 1 B
⑥ $\rightarrow \epsilon 1001\epsilon$ (~~using 0A~~ $B \rightarrow \epsilon$)
1001

R.M.

① $S \rightarrow A1B$
② $\rightarrow A1\underline{0}B$ (using $B \rightarrow 0B$)
③ $\rightarrow A100B$ (using $B \rightarrow 0B$)
④ $\rightarrow A1001B$ (using $B \rightarrow 1B$)
⑤ $\rightarrow A1001\epsilon$ (using $B \rightarrow \epsilon$)
⑥ $\rightarrow \epsilon 1001\epsilon$ (using $A \rightarrow \epsilon$)

1001

L.M.D → Left most derivation.

Ambiguity of Grammar → 2 or more Parse tree

→ Only one

Ambiguous Grammar / Unambiguous Gramma

* Ambiguous Grammar → A Grammar is said to be ambiguous if their exist more than one left most Derivations or more than one Right most Derivation or more than one parse tree for the given input string.

Let us consider a string w generated by the grammar w = id + id x id

Grammar

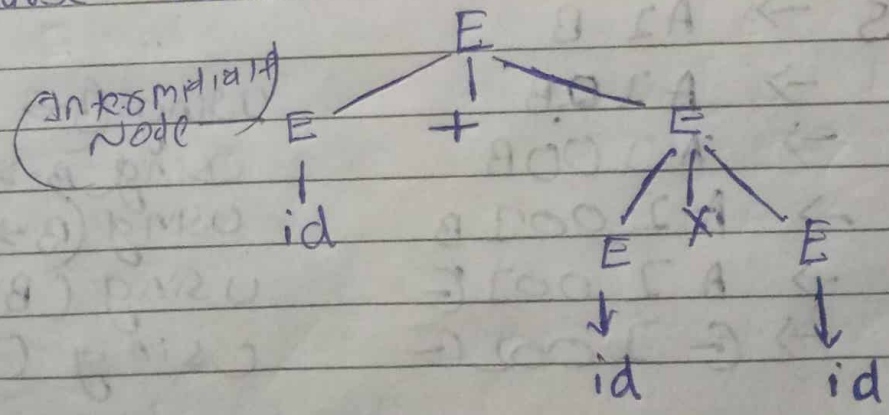
E → E+E / E x E / id

Reason 1

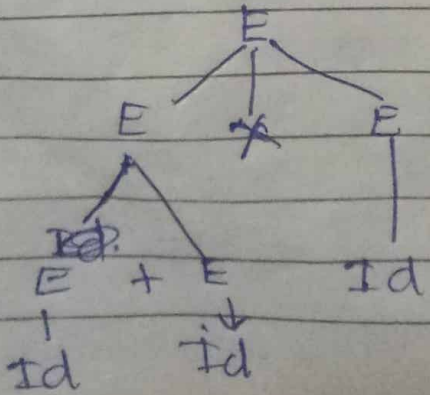
L.M.D for above example.

L.M.D

Parse Tree



Frist Tree



2nd Tree

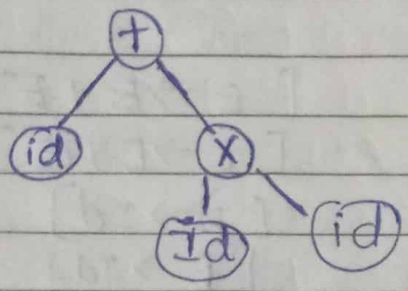
se tree

Syntax tree

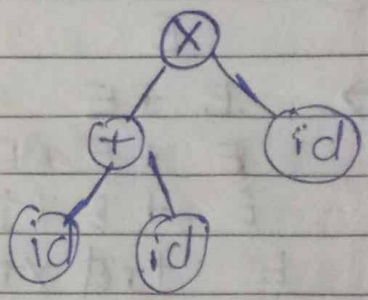
Since 2 parse tree exist ~~in L.M.D~~ for string w Therefore the grammar is ambigious.

Reason: 02

Let us draw the syntax trees for the String w



Syntax tree (I)



Syntax Tree-(2)

Again 2 Trees are formed for same Grammar.

So Ambig

Reason03

Let us consider a string w generated by the grammar w = id + id * id

First write down L.M.d.

L.M.D (1)

$E \rightarrow E+E$

$E \rightarrow E + E$ . . Using $E \rightarrow E+E$

$\rightarrow id + E$ Using $E \rightarrow id$

$\rightarrow id + E * E$ Using $E \rightarrow E*E$

$\rightarrow id + id * E$ Using $E \rightarrow id$

$\rightarrow id + id * id$ Using $E \rightarrow id$

L.M.D ②

E

$E \rightarrow E + E$

$\rightarrow E + E * E$

$\rightarrow id + E * E$

$\rightarrow id + id * E$

$\rightarrow id + id * id$.

US

$[E \rightarrow E + E]$

$[E \rightarrow E * E]$

$[E \rightarrow id]$

$[E \rightarrow id]$

$[E \rightarrow id]$

$[E \rightarrow id]$

R.M.D ①

E

$E \rightarrow E + E$

$E + E * E$

$E + E * id$

$E + id * id$

$id + id * id$

$[E \rightarrow E + E]$

$[E \rightarrow E * E]$

$[E \rightarrow id]$

$[E \rightarrow id]$

$[E \rightarrow id]$

R.M.D ②

E

$E \rightarrow E * E$

$E * id$

$E + E * id$

$E + id * id$

$id + id * id$

$[E \rightarrow E * E]$

$[E \rightarrow id]$

$[E \rightarrow E + E]$

$[E \rightarrow id]$

$[E \rightarrow id]$

Since 2 R.M.D & 2 L.M.D exist

For string w

∴ The gramar is ambiguous

Example 2

$S = aSb \mid SS$

$S = \epsilon$

For the string aabb the above gramar genrates two parse tree check whether the given Grammar is ambiguous or not.

$w = aabb$

LSB (1)

RSB (1)

$S \rightarrow aSb$

$a\,aSbb$

$a\,a\,bb$

E $S \rightarrow aSb$

$S \rightarrow aSb$

$S \rightarrow \epsilon$

LSB (2)

$S \rightarrow aSb$

$aSSb$

$aaSbb$

$a\,a\,bb$

$S \rightarrow aSb$

$S \rightarrow SS$

$S \rightarrow aSb$

$S \rightarrow \epsilon$

LSB (3)

$S \rightarrow aSb$

$\rightarrow aSSb$

$\rightarrow a\epsilon Sb$

$\rightarrow aaSbb$

$\rightarrow aabb$

$S \rightarrow aSb$

$S \rightarrow SS$

$S \rightarrow \epsilon$

$S \rightarrow aSb$

RSb (2)

S → aSb → aSSb → aSaSbb → aSaεbb → aεaεbb

[S → aSb]
[S → SS]
[S → aSb]
[S → ε]
[S → ε]

So

aabb

RSb (3)

~~S → aSb~~

a SSb

aS aSbSb

aa

RSb (3)

S → aSb
aSSb
aSεb
aaSbb
aabb

(S → aSb)
(S → SS)
(S → ε)
(S → aSb)