CONTEXT FREE GRAMMARS

* Grammar :

- Grammar is a set of formal rules which check the correctness of the sentences or we can construct the correct sentences using these rules. :
- These sentences are known as grammatical sentenc and the rules as syntax. The meaning is semantic
- Machine is a formal system which recognizes as we as generates the class of language.
- These are formal languages and we want to have a relation between the language and the machine.
- Two as pects of language
  i) Generative Capacity
  ii) Grammatical Constituents.
- Generative Capacity indicates that the grammar should generate all and only the sentences belonging to the language whereas Grammatical constituents gives H stress on primitive structures which compose the senten
- The first will consider the consistency and completenes of the formal system where as second deals with synthesis and analysis.

Grammar Def^n:

- Grammar is used to specify syntax for language. The grammar will generate the strings.
  It is a quadruple and is defined as:

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

Where, V = A finite set of Variables / Non-terminals
T = Finite set of terminals
P = Set of production / rewriting rules
S = start variable.

Variables → represented by capital letters
Terminals → represented by small letters / operators

e.g: $G = (\{E\}, \{+, *, id\}, P, E)$
- $\{E\}$ → Variables
- $\{+, *, id\}$ → Terminals
- P → Production
- E → start symbol

P : $E \to E+E$
$E \to E*E$
$E \to id$

OR $E \to E+E \mid E*E \mid id$

Derivation of a sentences (word)
↳ (string of terminals)

e.g: "id + id", derive it.

$E \Rightarrow E+E$ (using $E \to E+E$)
$\Rightarrow id+E$ (using $E \to id$)
$\Rightarrow id+id$ (using $E \to id$)

Terminals

* Context Free Grammar : (CFG)

- Context free Grammar generates context free language (CFL) that are recognized by push down automata (PDA)
- A grammar is said to be context free Grammar if the productions are of the form,

$$A \to \alpha$$

i.e. (one non terminal) = Finite string of terminals and non-terminals (sentential form)

where,
A → non-terminal
α → some sentential form (Combination of variable & terminals)

Here, the left hand side of the production can contain only one non-terminal and right hand side of the production can contain combination of non-terminal and terminals. The start variable can be present on the right hand side of the production.

e.g: $G = (\{S, A, B\}, \{a, b\}, P, S)$

where P:

$S \rightarrow aA \mid bB \mid a$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

* Variables:

Symbols that are taking part in derivation process but are not part of the derived string.

* Terminals:

Symbols that are present in the derived sentences.

* Derivation:

The sequence of production used to derive the sentence is known as derivation.

* Derivation of Grammar:

- A string which is generated from the given product rules and represented in the form of a tree, such tree is known as derivation tree or parse tree or Rule tree.
- The derivation tree will have labels to all the nodes. These labels could be non terminal symbols or terminal symbols or ε.
- The root is always labelled start.
- Nodes having labels as non-terminal are the interior nodes.
- All the leaf nodes will be terminal symbols.
- If a particular non terminal is generating a string of terminals and non-terminals of length n then that will have n children.
These children will be attached from left to right in sequence.

(1) Left Most Derivation (LMD)

If at each step in the derivation process, we replace the leftmost variable by its production then the derivation is said to be leftmost derivation.

(2) Right Most Derivation (RMD)

If at each step in the derivation process, we replace the rightmost variable by its production then the derivation is said to be right most derivation.

* Generation of CFG:

r	L(r)	CFG
a	{a}	S → a
b	{b}	S → b
a|b	{a} ∪ {b} = {a, b}	S → a | b
a^*	{ϵ, a, aa, ----}	S → aS | ϵ
a^+	{a, aa, ----}	S → aS | a
a·b	{ab}	S → ab
$(ab)^*$	{ϵ, ab, abab, ---}	S → abS | ϵ
$(a+b)^*$	{ϵ, a, b, aa, ab, ba, --}	S → aS | bS | ϵ

Q. Write CFG for
i) $r = (a|b)^*$
ii) $r = ba^*$
iii) $r = b^* a^*$
iv) $r = (baa + abb)^*$

→

i) $r = (a|b)^*$ ∴ G = ({S}, {a}, {S → aS|bS|ϵ}, S)
P: S → aS|bS|ϵ

ii) $r = ba^*$
S → bA
A → aA | ϵ
∴ G = ({S, A}, {a, b}, {S → bA, A → aA|ϵ}, S)

iii) $r = b^* a^*$ ∴ G = ({S, A, B}, {a, b}, {S → BA, B → bB|ϵ, A → aA|ϵ}, S}
S → BA
B → bB | G
A → aA | ϵ

* Ambiguous Context free Language :

→ The Grammar of a language is said to be Ambiguous if the same string can be generated in two different ways. (using more than one LMD/RMD)

→ To determine that the given grammar is ambiguous we need to desire at least one sentence using more than one LMD or RMD.

Q. check whether following grammar is ambiguous or not.

** $E \rightarrow E+E \mid E*E \mid id$

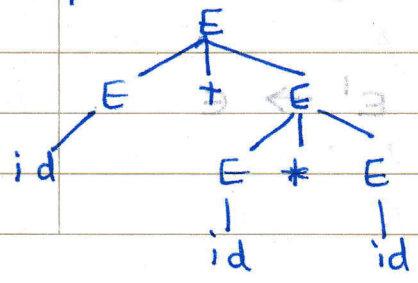
→ LMD1: Deriving LMD for sentence id + id * id

$E \Rightarrow E+E$ (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 $E \rightarrow id$)

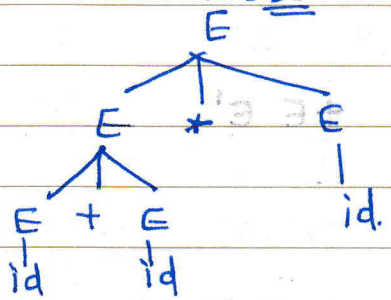
LMD 2 :

$E \Rightarrow E*E$ (using $E \rightarrow E*E$)
$\Rightarrow E+E*E$ (using $E \rightarrow E+E$)
$\Rightarrow id+E*E$ (using $E \rightarrow id$)
$\Rightarrow id+id*E$ (using $E \rightarrow id$)
$\Rightarrow id+id*id$ (using $E \rightarrow id$)

parse tree for LMD1



LMD2



As there is no unique parse tree for generating a string, the grammar is said to be ambiguous.

* Unambiguous grammar
- Grammar is unambiguous if it can derive all sentences using exactly one LMD / RMD.

* Ambiguity Resolution :
→ To resolve the ambiguity, no method exists
∴ Take a heuristic approach.

Rule 1 :

Let $A \to A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \beta_3 \mid \dots \mid \beta_n$

Then, $A \to \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_n A'$

$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_n A' \mid \epsilon$

Q. Eliminate the ambiguity in the given grammar

$E \to E + E \mid E * E \mid id$

→

$E \to E + E \mid E * E \mid id$

$E \to A$, $E \to A$, $+E \to \alpha_1$, $E \to A$, $*E \to \alpha_2$, $id \to \beta_1$

Rule 1: $E \to id\ E'$ ($E \to A$, $id \to \beta_1$, $E' \to A'$)

Rule 2: $E' \to +E\ E'$ ($E' \to A'$, $+E \to \alpha_1$, $E' \to A'$)

Rule 3: $E' \to *E\ E'$ ($E' \to A'$, $*E \to \alpha_2$, $E' \to A'$)

Rule 4: $E' \to \epsilon$ ($E' \to A'$)

* Simplification of CFG

* Elimination of Useless Symbols :

- **Useless Symbol :**

A symbol 'X' is useful if

$$S \xRightarrow{*} \alpha X \beta \xRightarrow{*} \omega$$

where,

$S \rightarrow$ start symbol

$\alpha, \beta \rightarrow$ sentential forms

$\omega \rightarrow$ sentence

Otherwise 'X' is useless

- A production where useless symbols are present is known as useless production.

- **Elimination procedure :**

Given CFG $G = (V, T, P, S)$

Define $G' = (V', T', P', S)$ be the CFG with no useless production (symbol) such that $L(G') = L(G)$

Step 1 : Initialize P' to P

step 2 : Find useless symbols

* Symbol is useless if it is not reachable from start symbol.
* Symbol is useless if it cannot derive any sentence (i.e. any combination of terminals)

step3 : Eliminate all occurrence of useless symbols produ

* Elimination of Unit Production :

- **Unit production :**

  A production of the form $A \rightarrow B$ where A & B are variables is called as Unit productions.

Note :

Chain of unit production is :

$A \rightarrow X_1 \rightarrow X_2 \rightarrow \text{----} \rightarrow X_n \rightarrow B$

- **Elimination procedure :**

Given CFG, $G = (V, T, P, S)$

Define $G' = (V', T', P', S)$ be the CFG with no unit productions such that $L(G') = L(G)$.

step 1 : Initialize P' to P.

step 2 : Find unit productions.

Say $A \rightarrow B$ is a unit production of A and $B \rightarrow \beta_1 | \beta_2 | \text{---} | \beta_n$ are non-unit productions of B, then add non-unit productions of B to A (if not present)

i.e. $A \rightarrow \beta_1 | \beta_2 | \text{----} | \beta_n$.

step 3 : Delete all unit productions.

- **Non - unit productions :**

A production which is not of the form $A \rightarrow B$ where A & B are variables is called as non-unit productions.

* Elimination of NULL production :

- **NULL production :**

A production of the form $X \to \epsilon$ or $X \to \Lambda$, where X is a variable is called NULL production

- **Nullable Variable :**

A variable 'X' is said to be nullable if $X \to \epsilon$ or $X \overset{*}{\Rightarrow} \epsilon$

- **Elimination Procedure :**

Given CFG, $G = (V, T, P, S)$
Define $G' = (V', T', P', S)$ be the CFG with no null productions such that $L(G') = L(G) - \{\epsilon\}$

step 1 : Initialize P' to P.
step 2 : Find the nullable variables :
Say $A \to \alpha$ is a production where α contains some nullables variables, then add to A, the productions obtained by deleting all possible subsets of nullable variables from α (if not present)
step 3 : Delete null productions.

Page No.:

Date:

NORMAL FORMS

→ The normal forms are used to recorite the CFG as per the specified conditions. When certain condition is imposed on the RHS, the grammar is said to be in the Normal Form.

* Chomsky Normal Form (CNF)

- Any ε-free CFL can be generated by a CFG whose productions are of the form

  $A \rightarrow BC$ or $A \rightarrow a$

  where, A, B & C are variables and a is a terminal

- In the Chomsky's Normal Form there will be restriction on the RHS.

* Conversion procedure from CFG to CNF:

1) Simplify the given CFG by eliminating Null, Unit and Useless productions.
2) Add to the solution the productions that are already in CNF.
3) For the non-CNF productions:
   a) Replace the terminals by some variables.
   b) Limit the no. of variables on RHS.

* Greibach Normal Form (GNF):

- Any ϵ-free CFL can be generated by a CFG whose productions are of the form

$$\boxed{A \to a\gamma}$$

where,

$A \to$ a variable
$a \to$ a terminal
$\gamma \to$ string of 0 or more variables which can be empty

* Conversion procedure for CFG to GNF:

1) Simplify the given CFG by eliminating Null, unit and useless productions.
2) Use any combination of Rule1 and Rule2 to get the CFG in GNF.

Rule1: Let $A \to B\alpha$ be some A-productions and $B \to \beta_1 | \beta_2 | \cdots | \beta_n$ be B-productions then we can write A-productions as,

$$A \to \beta_1\alpha | \beta_2\alpha | \cdots | \beta_n\alpha$$

Rule2: Let $A \to A\alpha_1 | A\alpha_2 | \cdots | A\alpha_r$ be A-productions and $A \to \beta_1 | \beta_2 | \cdots \beta_s$ be the remaining A-productions then introduce a new variable B, then

B-productions : $B \to \alpha_i | \alpha_i B$, $1 \le i \le r$
A-productions : $A \to \beta_i | \beta_i B$, $1 \le i \le s$

Note: Let $A \to aEFba$, then we can rewrite A as:
$A \to aEFX_1X_2$ (X_1 & X_2 → auxiliary variables),
$X_1 = b$, $X_2 = a$

Page No. :

Date :

CHOMSKY'S HIRARCHY

- Regular expression language, Context free language are few example of languages.
- These languages along with some new classes of languages were classified by Noam Chomsky.
- According to Noam Chomsky, these are four types of grammars as follows:

Type 0 → Unrestricted Grammar / Semi Thue Grammar / phrase Structure Grammar
Type 1 → Context Sensitive Grammar
Type 2 → Context Free Grammar
Type 3 → Regular Grammar

1) Type 0 → Unrestricted Grammar

→ The grammar is also known as Phrase Structure grammar / Semi Thue Grammar.
→ The largest class defined in CHOMSKY'S HIRARCHY is TYPE 0 Grammar.
→ The productions in type 0 grammar are of the form $\alpha \rightarrow \beta$ where α & β are strings formed by the symbols in the grammar and also α is not NULL.
→ There are no restrictions on the production rules in this case.
→ It generates Recursively Enumerable Language. (REL).
→ The Language generated (strings) will be recognized by the Turing Machine. Hence a TM can be designed such that its algorithmic way can check the acceptance

of any strings belonging to any 'Type 0' grammar.

e.g.: Let, $V = \{A, B, S\}$

$T = \{a, b, c\}$

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

P: $S \rightarrow SA$

$SA \rightarrow AB$

$A \rightarrow a$

2> Type 1 : Context Sensitive Grammar (CSG)

→ It is a grammar in which the productions are of the form $\alpha \rightarrow \beta$ such that the length of the string β will be atleast as long as the length of α.

→ It means, the left hand side of each production is never longer than the Right hand side.

$|\beta| = |\alpha|$

→ It generates context sensitive language (CSL).

→ The generated languages are recognized by linear bounded automata (LBA).

→ The productions of the form

$\alpha P \beta \rightarrow \alpha Q \beta$ where, $P \in$ non-terminal

$\alpha, \beta, Q \in \{T \cup N\}^*$ (string of terminal & nonterminal)

→ The replacement of P by Q is possible only if it is preceded by α and is followed by β. Hence the context in which P is appearing will be important for replacement. If the context is not appropriate the rule will not be applied.

→ e.g.: Let $G = (\{S, A, B\}, \{a, b\}, P, S)$

where, P: $S \rightarrow aAb$

$aA \rightarrow aB$

$aA \rightarrow aaA$, $A \rightarrow b$, $A \rightarrow a$

* ~~Context~~ Type 2 → Context Free Grammar (CFG):

→ CFG generates context free languages (CFL).
→ The languages generated are recognized by push down automata (PDA).
→ A grammar is said to be CFG, if the productions are of the form,

$$A \to \alpha$$

i.e. (one non-terminal) → finite string of sentential form.
where,

A → non-terminal
α → some sentential form $(V+T)$

→ Here, the LHS of the production can contain only one non-terminal and RHS. of the production can contain combination of non-terminal and terminal. The start variable can be present on the RHS of the production.
→ e.g:

$G = (\{S, A, B\}, \{a, b\}, P, S)$

where P:

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

* Type 3 → Regular Grammar (RG)

→ If the production rules are of the form in which the RHS contains at the most one non-terminal symbol, the grammar is known as regular grammar.
→ Regular grammar generates regular language (RL).

→ The generated languages are recognized by FSM.

→ These regular languages can also be expressed by simpler expressions called as Regular Expression (RE).

→ The production is of the form,

only one nonterminal

$S \rightarrow sT^*/T^*$ or $S \rightarrow T^*s/T^*$

where, S is non Terminal,
T is any combination of terminals.

→ The grammar is said to be left linear if the productions contain any one nonterminal symbol which occurs at the leftmost.

e.g: $S \rightarrow Sa$
$S \rightarrow Pb$
$P \rightarrow b$

→ The grammar is said to be right linear if the productions contain any one nonterminal symbol which occurs at the rightmost

e.g: $S \rightarrow as$
$S \rightarrow bP$
$P \rightarrow a$

→ The grammar is said to be linear if any non-terminal generates a single nonterminal on the RHS.

→ The grammar are said to be equivalent if the strings generated by them are identical.

source: Theory o
Computat" - Kav

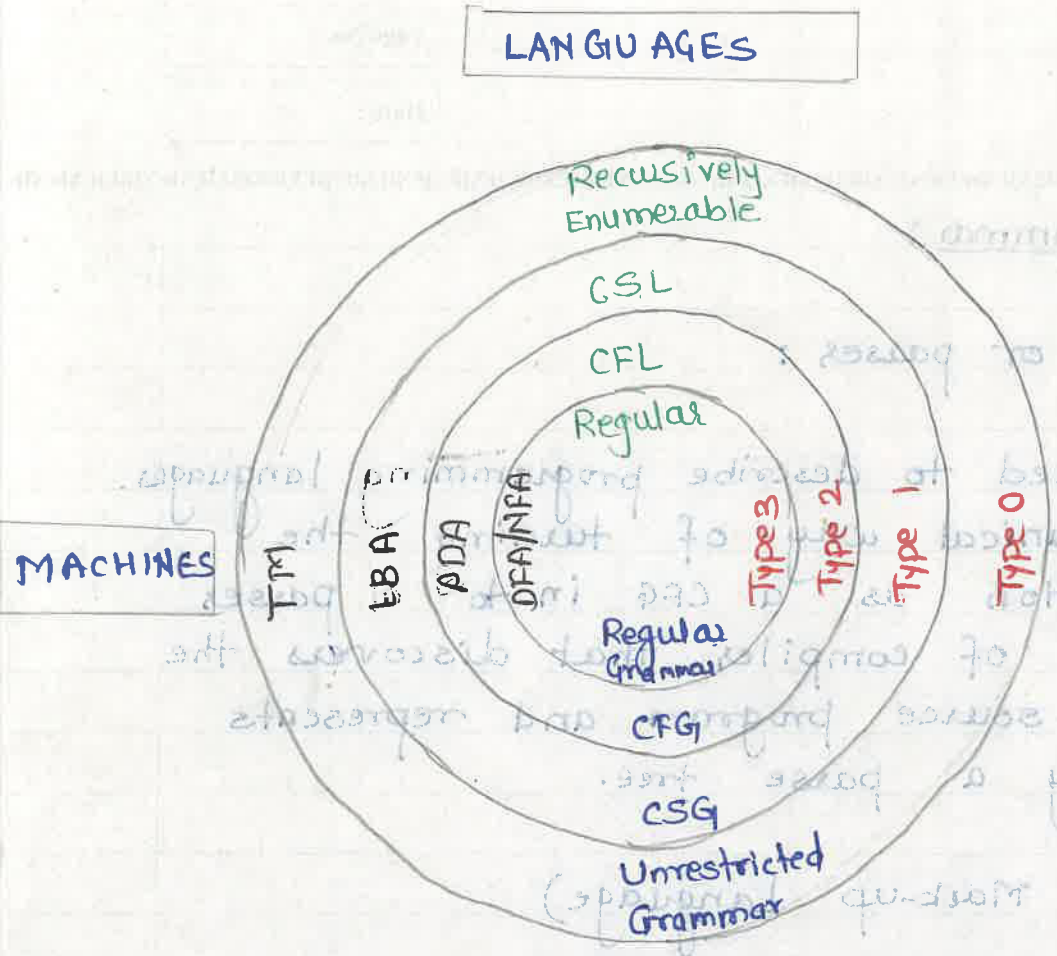


Fig: Chomsky Hierarchy of formal languages

Sr. No.	Grammar Type	Grammar accepted	Language accepted	Automation
1.	Type 0	Unrestricted grammar	Recursively Enumerable Language (REL)	Turing Machi (TM)
2.	Type 1	Context Sesitive Gramma (CSG)	Context Sensitive Language (CSL)	Linear Bounded Automata (LBA)
3.	Type 2	Context free grammar (CFG)	Context Free Language (CFL)	Pushdown Automa (PDA)
4.	Type 3	Regular grammar	Regular Language	Finite Automat (FA)

* Applications of Grammar :

1) Syntax Analyzer or parser :

→ Grammars are used to describe programming languages.
→ There is a mechanical way of turning the language description as a CFG in to a parser is a component of compiler that discovers the structure of the source program and represents that structure by a parse tree.

2) XML (Extensible Mark-up Language)

→ An essential part of XML is DTD (Document Type Definition) which is essentially a CFG that describes the allowable tags ~~that~~ and the ways in which these tags may be nested.