INDEX

Grammar: Grammar is a set of rules used to define a valid sentence in any language.

Mathematical representation of grammar: Grammar is defined as a 4-tuple G={V,T,P,S} where

V - finite set of non-terminals
T - finite set of terminals
P – finite set of production rules
S – Start symbol

"Non terminals" are represented by capital letters A, B, C,… so on.
"Terminals" are represented by small letters a, b, c,… so on.
"Production rules" are of the form $\alpha \rightarrow \beta$, where

α contains non-terminal symbols or may also contain both terminals and non-terminals with atleast one non-terminal.
β contains terminal, non-terminal symbols or any combination of both terminals and non-terminals.

Formal Grammar: Grammar is an useful model when designing software that processes data with recursive structure. i.e., It denotes the structure of data.

Formal Language: Formal language is an abstraction for general characteristics of programming languages.

Classification of Grammars:

Chomsky Hierarchy: Noam Chomsky classified grammar into 4 types depending on production rules (.

1. Type 0 – Unrestricted Grammar
2. Type 1 – Context Sensitive Grammar (CSG)
3. Type 2 – Context Free Grammar (CFG)
4. Type 3 – Regular Grammar

Type 0 Grammar: It is also known as "unrestricted grammar" or "phase structured grammar". An unrestricted grammar is a formal grammar in which no restrictions are made on LHS and RHS of production.

Mathematical representation of Type 0 grammar: Type 0 Grammar is defined as a 4-tuple G={V,T,P,S} where

V - finite set of non-terminals
T - finite set of terminals
P – finite set of production rules of the form $\alpha A\beta \rightarrow \alpha\gamma\beta$
Where α – left context, β – right context,
$\alpha, \beta \in (V \cup T)^+$ and $\gamma \in (V \cup T)^*$,
i.e., no epsilon (ε) productions are allowed on LHS but epsilon (ε) productions are allowed on RHS.
S – Start symbol

The language that can be generated by making use of type 0 grammar is "recursive enumerable language". The machine format that is used to recognize strings of that particular grammar is "turing machine".

E.g: S→aSBA | abA
AB→BA
bB→bb
bA→ba
aA→aa

Type 1 Grammar: It is also known as "Context Sensitive Grammar (CSG)". Context Sensitive Grammar (CSG) is a formal grammar in which LHS and RHS of any production rule may be surrounded by a context of terminals and non-terminals.

Mathematical representation of Type 1 grammar: Type 1 Grammar is defined as a 4-tuple G={V,T,P,S} where

V - finite set of non-terminals
T - finite set of terminals
P – finite set of production rules of the form $\alpha A\beta \rightarrow \alpha\gamma\beta$
Where α – left context, β – right context,
$\alpha, \beta \in (V \cup T)^+$ and $\gamma \in (V \cup T)^+$
i.e., no epsilon (ε) productions are allowed on LHS and RHS.
S – Start symbol

The language that can be generated by making use of type 1 grammar is "Context Sensitive language (CSL)". The machine format that is used to recognize strings of that particular grammar is "Linear Bounded Automata (LBA)".

E.g: S→abc | aSBC
CB→BC
bB→bb

Type 2 Grammar: It is also known as "Context Free Grammar (CFG)". Context Free Grammar (CFG) is a formal grammar in which LHS and RHS of any production rule is free from a context of terminals and non-terminals.

Mathematical representation of Type 2 grammar: Type 2 Grammar is defined as a 4-tuple G={V,T,P,S} where

V - finite set of non-terminals
T - finite set of terminals
P – finite set of production rules of the form $A \rightarrow \gamma$
Where and $\gamma \in (V \cup T)^*$,
i.e., $A \rightarrow (V \cup T)^*$,
i.e., LHS of any production rule contains only a single non-terminal and epsilon (ε) productions are allowed on RHS.
S – Start symbol

The language that can be generated by making use of type 2 grammar is "Context Free language (CFL)". The machine format that is used to recognize strings of that particular grammar is "Push Down Automata (PDA)".

E.g: S→aSb
S→ab

Type 3 Grammar: It is also known as "Regular Grammar (RG)". Regular Grammar (RG) is a formal grammar with restrictions on both LHS and RHS. i.e., LHS contains only a single non-terminal and RHS may contain single terminal (or) single terminal followed by non terminal (or) nothing.

Mathematical representation of Type 3 grammar: Type 3 Grammar is defined as a 4-tuple G={V,T,P,S} where

V - finite set of non-terminals
T - finite set of terminals
P – finite set of production rules of the form $A \rightarrow t \mid tB \mid \varepsilon$,
Where t – terminals
A,B – non terminals
S – Start symbol

The language that can be generated by making use of type 3 grammar is "Regular language (RL)". The machine format that is used to recognize strings of that particular grammar is "Finite Automata (FA)".

E.g: S→aS
S→a

Types of formal languages: Based on properties of Chomsky hierarchy of grammars, formal languages are classified into 4 types.

1. Regular Language
2. Context Free Language
3. Context Sensitive Language
4. Recursive Language and Recursive Enumerable Language

Regular Language: It is a formal language generated by regular grammar.

Properties of Regular Language:

1. It is accepted by DFA, NFA and read only turing machine.
2. It is described by regular expressions.
3. It is generated by regular grammar.

Closure Properties of Regular Language:

Two regular languages are closed under the following.

1. Union
2. Concatenation
3. Complementation
4. Intersection
5. Kleene closure
6. Transpose
7. Substitution
8. Homomorphism
9. Inverse homomorphism

Methods used to prove the given language is not regular – pumping lemma.

E.g: $L=\{a^n \mid n\geq 1\}$ is a regular language generated by regular grammar
S→aS
S→a

Context Free Language: It is a formal language generated by context free grammar.

Properties of Context Free Language:

1. It is accepted by pushdown automata.
2. It is generated by context free grammar.
3. It doesn't satisfy operations such as intersection, complement, difference.

Closure Properties of Context Free Language:

Two context free languages are closed under the following.

1. Union
2. Concatenation
3. Kleene closure
4. Transpose
5. Substitution
6. Homomorphism
7. Inverse homomorphism

Methods used to prove the given language is not context free – pumping lemma.

E.g: $L=\{a^n b^n \mid n\geq 1\}$ is a context free language generated by context free grammar
S→aSb
S→ab

Context Sensitive Language: It is a formal language generated by context sensitive grammar.

Properties of Context Sensitive Language:

1. It is accepted by pushdown automata.
2. It is generated by context sensitive grammar.
3. It doesn't satisfy operations such as complementation, difference.

Closure Properties of Context Sensitive Language:

Two context sensitive languages are closed under the following.

1. Union
2. Concatenation
3. Kleene closure
4. Intersection
5. Transpose

Methods used to prove the given language is not context sensitive – pumping lemma.

E.g: $L=\{a^n b^n c^n \mid n\geq 1\}$ is a regular language generated by context sensitive grammar

$S\rightarrow abc \mid aSBC$

$CB\rightarrow BC$

$bB\rightarrow bb$

Recursive Language: Any language L over alphabet $\sum$ is said to be recursive, if there is a turing machine that accepts every word present in the language L and rejects every word which is not in L.

Properties of Recursive Language:

1. It is accepted by turing machine.
2. It is generated by unrestricted grammar.

Closure Properties of Recursive Language:

Two recursive languages are closed under the following.

1. Union
2. Concatenation
3. Complementation
4. Intersection
5. Kleene closure

Recursive Enumerable Language: A recursive enumerable language L over alphabet $\sum$ accepts every word in L of turing machine M (or) either rejects or loops every word which is not in L.

$$\text{Accept}=L$$

$$\text{Reject} + \text{Loop} = \bar{L}$$

i.e., a turing machine on same input can either accept a language or else, will run in loop forever.

Properties of Recursive Enumerable Language:

1. It is accepted by turing machine.
2. It is generated by unrestricted grammar.

Closure Properties of Recursive Enumerable Language:

Two recursive enumerable languages are closed under the following.

1. Union
2. Concatenation
3. Intersection
4. Kleene closure

E.g: $L=\{a^n b^n a^n \mid n\geq 0\}$ is a recursive enumerable language generated by unrestricted grammar $S\rightarrow aSBA \mid abA$

$AB\rightarrow BA$

$bB\rightarrow bb$

$bA\rightarrow ba$

$aA\rightarrow aa$

Context Free Grammar:

A grammar G=(V,T,P,S) is said to be context free grammar if it is recursive and free from context. i.e., productions are in the form of A→ $(V \cup T)^*$ in which LHS of any production rule contains only a single non-terminal and epsilon (ε) productions are allowed on RHS.

CFG is formally defined as G=(V,T,P,S) where

V - finite set of non-terminals
T - finite set of terminals
P – finite set of production rules of the form A→ $(V \cup T)^*$,
S – Start symbol

Backus naur Form:

It was proposed by John Backus and Peter Naur. If there are many productions with same LHS variable, then they can be combined into single production LHS → RHS, where

LHS is common variable of LHS of productions
LHS is RHS productions separated by vertical bar (|)
E.g: S→aS
S→ε

ε should be written at the end. Therefore, S→aS | ε

Derivation:

The process of generating a string from the production rule of a grammar is called derivation.

Types of Derivation:

1. Leftmost derivation
2. Rightmost derivation

Leftmost derivation:

A derivation is called leftmost derivation, if leftmost variable is replaced with some production rule of a grammar.

E.g: S→aSbS | ab
Let the string to be derived is aabbab.
LMD: S→a**S**bS
→aabb**S**
→aabbab

Rightmost derivation:

A derivation is called rightmost derivation, if rightmost variable is replaced with some production rule of a grammar.

E.g: S→aSbS | ab
Let the string to be derived is aabbab.
RMD: S→aSb**S**
→a**S**bab
→aabbab

Derivation tree:

The diagrammatical representation of the derivation process by a tree structure is known as "derivation tree".

It is also known as "parse tree" because it is created by parser (second phase of compiler) to check syntax of the statement.

It is also defined as an ordered tree in which LHS of production rule represents parent node and RHS of production rule represents children nodes.

Properties of Derivation tree:

In derivation tree of CFG,

1. Root node is represented starting variable.
2. Leaf nodes are represented by terminals or epsilon (ε).
3. Internal nodes are represented by non- terminals.
4. If $A \rightarrow \alpha_1\ \alpha_2\ \alpha_3 \ldots\ \alpha_n$ then A is parent of nodes $\alpha_1\ \alpha_2\ \alpha_3 \ldots\ \alpha_n$

Ambiguous Grammars:

If more than one parse tree (or) more than one LMD (or) more than one RMD is possible to derive the same string, then that grammar is said to be "Ambiguous grammar", Otherwise it is known as unAmbiguous grammar.

Yield of a derivation tree:

The string obtained by reading only the leaves of derivation tree from left to right (ε-symbols are ignored) is known as "yield" of a derivation tree.

Simplification of Context Free Grammars:

Purpose: CFG may contain different types of useless symbols, unit productions, null productions. These may increase number of steps in generating a language from CFG.

Therefore simplification of CFG will reduce variables and productions, so that time complexity for generating language become less.

Procedure to simplify CFG:

CFG can be simplified by performing the following steps:

1. Elimination of ε-Productions
2. Elimination of Unit Productions
3. Elimination of Useless Productions

Elimination of ε-Productions:

ε-Productions (null production): A production in the form of "A→ε" is called as null production.

Procedure to eliminate ε-Productions:

1. Identify ε-Productions in given CFG.(in the form of A→ε)
2. To eliminate A→ε, then write all the productions from CFG whose right side contains A.
3. Replace each occurrence of A with 'ε' to get production without 'ε'and eliminate A→ε.
4. The obtained productions in step3 must be added to the grammar.
5. A→ε is allowed only if A is not used on RHS of any production.

Elimination of unit-Productions:

Unit-Productions: A production in the form of "A→B" is called as unit production.

Procedure to eliminate Unit-Productions:

1. Identify unit-Productions in given CFG.(in the form of A→B)
2. To eliminate A→B, then write all the productions from CFG whose left side contains A.
3. Replace B with 'A'and eliminate A→B.
4. The obtained productions in step3 must be added to the grammar.

Elimination of useless-Productions:

Useless-Productions: Useless productions are of 2 types.

1. Non-generating productions
2. Non-reachable productions

Non-generating productions: Non-generating productions are the productions which do not produce terminal string. Non-generating symbols are the symbols because of which productions cannot produce terminal string.

Non-reachable productions: Non-reachable productions are the productions which cannot be reached from starting symbol.

Procedure to eliminate useless-Productions:

1. Identify non generating symbols in given CFG.
2. To eliminate non generating symbol A, then remove all the productions from CFG containing A on LHS or RHS.
3. Identify non reachable productions and remove them.
4. If start symbol is non generating symbol, it cannot be removed.
5. If symbol is non reachable but generating symbol, it cannot be removed.

Normalization of CFG:

The productions in CFG are of the form A→ $(V \cup T)^*$. i.e., there are no restrictions on RHS of CFG. There may be any number of terminals and non terminals in any combination.
If we made restrictions on RHS of CFG specifying that there should be fixed number of terminals and non terminals, it results in normal form.

Normal form:

If every production of CFG is converted into some standard form, grammar is said to be in normal form.

Use of normalization:

If some standard form is specified, then complexity of automata design will be less and easy to implement the automata.

Types of normal forms:

Normal forms are of 2 types. They are

1. Chomsky Normal Form (CNF)
2. Greibach Normal Form (GNF)

Chomsky Normal Form (CNF):

Let G be the CFG. The grammar G=(V,T,P,S) is said to be in CNF, if all productions are of the form A → BC (or) A → a,
where A, B, C ∈ V(non terminals) and a ∈ T (terminals).

i.e., if grammar is in CNF, RHS should contain only one terminal or two non terminals.

Conversion from CFG to CNF:

1. Simplify the given CFG – Eliminate ε-Productions, Unit Productions, and Useless Productions.
2. For the productions which are not in CNF, (having non terminals and terminals on RHS), replace each terminal with new non terminal.
3. For the productions which are not in CNF, (having more than 2 non terminals on RHS), replace 2 non terminals with new non terminal.
4. Repeat 2,3 steps until we get CNF grammar.

Greibach Normal Form (GNF):

1. Let G be the CFG. The grammar G=(V,T,P,S) is said to be in GNF, if all productions are of the form $A \rightarrow aB^*$,
2. where A, B ∈ V(non terminals) and a ∈ T (terminals).
3. i.e., if grammar is in GNF, RHS should contain single terminal followed by any number of non terminals.

Left Recursion:

A grammar is said to be left recursive, if it contains production of the form

$A \rightarrow A\alpha_1 | A\alpha_2 | A\alpha_3 | \ldots | A\alpha_n | \beta_1 | \beta_2 | \beta_3 | \ldots | \beta_n$

where RHS contains α_i starting with A and

β_i which does not starts with A

Elimination of Left Recursion:

Let $A \rightarrow A\alpha \mid \beta$ is the production having left recursion. To eliminate left recursion, introduce a new variable A' such that

$A \rightarrow \beta A'$

$A \rightarrow \alpha A' | \varepsilon$

Conversion from CFG to GNF:

1. Convert the given CFG into CNF.
2. Replace the names of variables with $A_1, A_2, A_3, \ldots A_n$ (indexed form).
3. Convert it into GNF using the following rules
   i. If production is of the form $A_i \rightarrow A_j A_k$, i>j, then convert it into $A_i \rightarrow A_j A_k$, i<j by substituting A_j with its RHS.
   ii. If production is of the form $A_i \rightarrow A_j A_k$, i=j, then eliminate left recursion and eliminate ε productions .
   iii. If production is of the form $A_i \rightarrow A_j A_k$, i<j, then convert substitute A_j with its RHS to obtain GNF grammar.

Pumping Lemma for context free languages:

Every language is not context free. To show that L is not context free, pumping lemma is used. Pumping means "generating". Pumping lemma means "it is a method of generating many input strings from a given string." i.e., it is used to break a given long input string into several substrings."

Statement of pumping lemma:

1. Assume that the given language L is context free accepted by pushdown automata M.

2. Choose a string z ϵ L such that |z| ≥ n for some n.
3. Break the string into 5 strings u,v,w,x,y i.e., z=uvwxy such that
   i. |vx| ≥ 1
   ii. |vwx| ≤ n
4. By using pumping lemma, pump v, x to the suitable integer 'i' such that $uv^iwx^iy \notin L$ for i≥0. This contradicts our assumption.
5. Therefore L is not context free.

Closure Properties of Context Free Languages:

Closure property: *A set is* closed *under an operation if applying that operation to elements of the set results in an element of the set.*

For example, the set of natural numbers is closed under addition and multiplication but not under subtraction or division because 2-3 is negative and 1/2 is not an integer. The set of integers is closed under addition, subtraction and multiplication but not under division.

Context free languages are closed under the following:
1. Union
2. Concatenation
3. Kleene closure
4. Transpose

Context free languages are not closed under the following:
1. Complementation
2. Intersection