College of Information Technology
Dept. Software

Summary of Grammar Types

Grammars use to generate sentences of a language and to determine if a given sentence is in a language. Formal languages, generated by grammars, provide models for programming languages (Java, C, etc) as well as natural language important for constructing compilers.

Definition:

Grammar is a finite set of formal rules for generating **syntactically correct** sentences or **meaningful correct** sentences. A Grammar can contain mainly two elements **- Terminal (T)** and **Non Terminal (N)**.

Terminal Symbols (T)

It is a portion of the sentence generated by using a grammar. It is denoted by using **small letters**, such as a, b, c, d, etc.

Non Terminal Symbols

It takes part in the formation of a sentence, but not part of it. It is denoted by using **capital letters**, such as **A, B, C, D**, etc. It is also called auxiliary symbols.

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

Non-Terminal symbols

Non-Terminal symbols

Start Symbol

Productions

Productions rules must be in the form:

$$\alpha \rightarrow \beta$$

We can summarize the difference between the types of grammar in table 1.

Table 1: Grammar Types

Type 0	Type 1	Type 2	Type 3
Unrestricted Grammar	Context Sensitive Grammar	Context free Grammar	Regular Grammar
$\alpha \rightarrow \beta$ $\alpha \in (N \cup T)^{+}$ $\beta \in (N \cup T)^{*}$	$\alpha \rightarrow \beta$ $\lvert\alpha\rvert \leq \lvert\beta\rvert$ $\alpha, \beta \in (N \cup T)^{+}$	$\alpha \rightarrow \beta$ $\beta \in (N \cup T)^{*}$	$\alpha \rightarrow \beta$ $\propto \rightarrow \beta w \mid w$ or $\propto \rightarrow w\beta \mid w$

Type 0 (Unrestricted Grammar or Phrase-Structure Grammar)

In automaton, Unrestricted Grammar or Phrase Structure Grammar is the most general in the Chomsky Hierarchy of classification. **Type 0** grammar, generally used to generate **Recursively Enumerable languages**. It is called **unrestricted** because no other restriction is made on this except each of their **left hand sides being non-empty**. The left hand sides of the rules can contain terminal and non-terminal, but the condition is **at least one of them must be non-terminal.**

A Turing Machine can simulate unrestricted grammar. Unrestricted grammar can always be found for the language recognized or generated by any **Turing Machine**.

Example:

$\mathbf{S \longrightarrow A \mid B}$

$\mathbf{A \longrightarrow aA \mid a}$

$\mathbf{B \longrightarrow bB \mid b}$

The above grammar uses to describe the language L(G) ={ $a^i + b^j \mid i, j \geq 1$}.

Type 1 (Context sensitive grammar)

The Context Sensitive Grammar is formal grammar in which the **left-hand sides and right-hand sides of any production rules may be surrounded by a context of terminal and non-terminal grammar**. It is **less general** than Unrestricted Grammar and **more general** than Context Free Grammar **(CFG).**

Example:

S $\longrightarrow$ SAS | a

aAa $\longrightarrow$ bba

The language which can be described by the above grammar is L(G) = {$(bb)^i a$ | i ≥ 0}.

Type 2 (Context free grammar)

A grammar is said to be context-free, if every production is in the form:

A → α

where, **A** is non-terminal symbol and **$\alpha \in (N \cup T)^*$.** The set of production rules describes all possible strings. In this type, productions are simple replacements.

Production rules

In Context-Free Grammar **(CFG),** all rules are **one to one, one-to-many** or **one-to-none**. The **left hand side** of the production rule is always a **non-terminal symbol**.

Example:

S $\longrightarrow$ XaaX

X $\longrightarrow$ aX | bX | ε

The grammar uses to describe the language L(G)={$(a+b)^i$ aa $(a+b)^j$ | i,j ≥ 0}.

Type 3 (Regular grammar)

If all production of a CFG are of the form **A $\longrightarrow$ wB** or **A $\longrightarrow$ w**, where **A** and **B** are variables and **w ϵ NT*,** then we say that grammar **is right linear**. If all production of a CFG are of the form **A $\longrightarrow$ Bw** or **A $\longrightarrow$ w**, we call it **left linear**.

A right or left linear grammar is called a **Regular Grammar (RG)**. Every **regular expression** can be represented by a **regular grammar**. As there is a **finite automaton** for every **regular expression**, we can generate **a finite automaton** for the **regular grammar**.

Example:

S $\longrightarrow$ 0A

A $\longrightarrow$ 10A | ε

The above grammar is a right-linear grammar which describes the language

$L(G)=\{0(10)^i \mid i \geq 0\}$.

The left-linear grammar which describes the same language L(G) is:

S $\longrightarrow$ S10 | 0

Derivation tree of a grammar:

- Represents the **language** using an **ordered rooted tree**.
- **Root** represents the **starting** symbol.
- **Internal vertices** represent the **nonterminal** symbol that arise in the production.
- **Leaves** represent the **terminal** symbols.
    - ► If the production $A \rightarrow w$ arises in the derivation, where w is a **word**, the vertex that represents A has as **children vertices** that represent each symbol in w, in order from **left to right**.

Language Generated by a Grammar

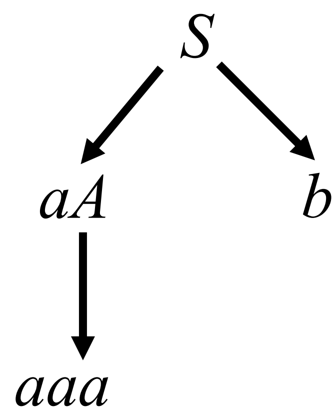
Example: Let $G = (\{S,A\},\{a,b\}, S,$
$\{S \rightarrow aA, S \rightarrow b, A \rightarrow aa\})$. What is $L(G)$?

Easy: We can just draw a tree
of all possible derivations.

- We have: $S \Rightarrow aA \Rightarrow aaa$.
- and $S \Rightarrow b$.

Answer: $L = \{aaa, b\}$.

Example of a *derivation tree* or *parse tree* or *sentence diagram*.



Example: Derivation Tree

Let G be a context-free grammar with the productions
$P = \{S \longrightarrow aAB,\ A \longrightarrow Bba,\ B \longrightarrow bB,\ B \longrightarrow c\}$.
The word w = acbabc can be derived from S as follows:

$$S \Rightarrow aAB \longrightarrow a(Bba)B \Rightarrow acbaB \Rightarrow acba(bB) \Rightarrow acbabc$$

Thus, the derivation tree is given as follows: