# Introduction to Grammars

In the literary sense of the term, grammars denote syntactical rules for conversation in natural languages. Linguistics have attempted to define grammars since the inception of natural languages like English, Sanskrit, Mandarin, etc.

The theory of formal languages finds its applicability extensively in the fields of Computer Science. **Noam Chomsky** gave a mathematical model of grammar in 1956 which is effective for writing computer languages.

Grammar

A grammar **G** can be formally written as a 4-tuple (N, T, S, P) where −

* **N** or **V*N*** is a set of variables or non-terminal symbols.
* **T** or **∑** is a set of Terminal symbols.
* **S** is a special variable called the Start symbol, S ∈ N
* **P** is Production rules for Terminals and Non-terminals. A production rule has the form α → β, where α and β are strings on V*N* ∪ ∑ and least one symbol of α belongs to VN.

Example

Grammar G1 −

({S, A, B}, {a, b}, S, {S → AB, A → a, B → b})

Here,

* **S, A,** and **B** are Non-terminal symbols;
* **a** and **b** are Terminal symbols
* **S** is the Start symbol, S ∈ N
* Productions, **P : S → AB, A → a, B → b**

Example

Grammar G2 −

(({S, A}, {a, b}, S,{S → aAb, aA → aaAb, A → ε } )

Here,

* **S** and **A** are Non-terminal symbols.
* **a** and **b** are Terminal symbols.
* **ε** is an empty string.
* **S** is the Start symbol, S ∈ N
* Production **P : S → aAb, aA → aaAb, A → ε**

Derivations from a Grammar

Strings may be derived from other strings using the productions in a grammar. If a grammar **G** has a production **α → β**, we can say that **x α y** derives **x β y** in **G**. This derivation is written as −

***x α y ⇒G x β y***

Example

Let us consider the grammar −

G2 = ({S, A}, {a, b}, S, {S → aAb, aA → aaAb, A → ε } )

Some of the strings that can be derived are −

S ⇒ aAb using production S → aAb

⇒ aaAbb using production aA → aaAb

⇒ aaaAbbb using production aA → aaAb

⇒ aaabbb using production A → ε

# Language Generated by a Grammar

The set of all strings that can be derived from a grammar is said to be the language generated from that grammar. A language generated by a grammar **G** is a subset formally defined by

L(G)={W|W ∈ ∑\*, S ⇒G **W**}

If **L(G1) = L(G2)**, the Grammar **G1** is equivalent to the Grammar **G2**.

### Example

If there is a grammar

G: N = {S, A, B} T = {a, b} P = {S → AB, A → a, B → b}

Here **S** produces **AB**, and we can replace **A** by **a**, and **B** by **b**. Here, the only accepted string is **ab**, i.e.,

L(G) = {ab}

### Example

Suppose we have the following grammar −

G: N = {S, A, B} T = {a, b} P = {S → AB, A → aA|a, B → bB|b}

The language generated by this grammar −

L(G) = {ab, a2b, ab2, a2b2, ………}

= {am bn | m ≥ 1 and n ≥ 1}

## Construction of a Grammar Generating a Language

We’ll consider some languages and convert it into a grammar G which produces those languages.

### Example

***Problem*** − Suppose, L (G) = {am bn | m ≥ 0 and n > 0}. We have to find out the grammar **G** which produces **L(G)**.

***Solution***

Since L(G) = {am bn | m ≥ 0 and n > 0}

the set of strings accepted can be rewritten as −

L(G) = {b, ab,bb, aab, abb, …….}

Here, the start symbol has to take at least one ‘b’ preceded by any number of ‘a’ including null.

To accept the string set {b, ab, bb, aab, abb, …….}, we have taken the productions −

S → aS , S → B, B → b and B → bB

S → B → b (Accepted)

S → B → bB → bb (Accepted)

S → aS → aB → ab (Accepted)

S → aS → aaS → aaB → aab(Accepted)

S → aS → aB → abB → abb (Accepted)

Thus, we can prove every single string in L(G) is accepted by the language generated by the production set.

Hence the grammar −

G: ({S, B}, {a, b}, S, { S → aS | B , B → b | bB })

### Example

***Problem*** − Suppose, L (G) = {am bn | m > 0 and n ≥ 0}. We have to find out the grammar G which produces L(G).

***Solution*** −

Since L(G) = {am bn | m > 0 and n ≥ 0}, the set of strings accepted can be rewritten as −

L(G) = {a, aa, ab, aaa, aab ,abb, …….}

Here, the start symbol has to take at least one ‘a’ followed by any number of ‘b’ including null.

To accept the string set {a, aa, ab, aaa, aab, abb, …….}, we have taken the productions −

S → aA, A → aA , A → B, B → bB ,B → λ

S → aA → aB → aλ → a (Accepted)

S → aA → aaA → aaB → aaλ → aa (Accepted)

S → aA → aB → abB → abλ → ab (Accepted)

S → aA → aaA → aaaA → aaaB → aaaλ → aaa (Accepted)

S → aA → aaA → aaB → aabB → aabλ → aab (Accepted)

S → aA → aB → abB → abbB → abbλ → abb (Accepted)

Thus, we can prove every single string in L(G) is accepted by the language generated by the production set.

Hence the grammar −

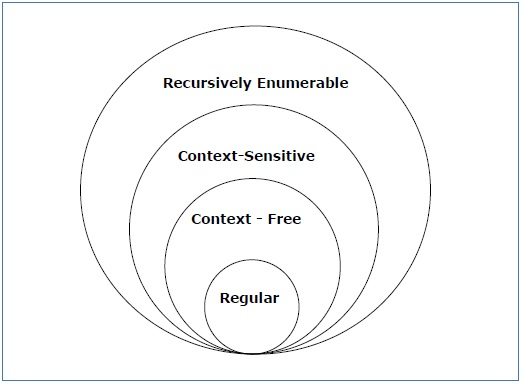
G: ({S, A, B}, {a, b}, S, {S → aA, A → aA | B, B → λ | bB })

# Chomsky Classification of Grammars

According to Noam Chomosky, there are four types of grammars − Type 0, Type 1, Type 2, and Type 3. The following table shows how they differ from each other −

| **Grammar Type** | **Grammar Accepted** | **Language Accepted** | **Automaton** |
| --- | --- | --- | --- |
| Type 0 | Unrestricted grammar | Recursively enumerable language | Turing Machine |
| Type 1 | Context-sensitive grammar | Context-sensitive language | Linear-bounded automaton |
| Type 2 | Context-free grammar | Context-free language | Pushdown automaton |
| Type 3 | Regular grammar | Regular language | Finite state automaton |

Take a look at the following illustration. It shows the scope of each type of grammar −



Type - 3 Grammar

**Type-3 grammars** generate regular languages. Type-3 grammars must have a single non-terminal on the left-hand side and a right-hand side consisting of a single terminal or single terminal followed by a single non-terminal.

The productions must be in the form **X → a or X → aY**

where **X, Y ∈ N** (Non terminal)

and **a ∈ T** (Terminal)

The rule **S → ε** is allowed if **S** does not appear on the right side of any rule.

Example

**X → ε**

**X → a | aY**

**Y → b**

Type - 2 Grammar

**Type-2 grammars** generate context-free languages.

The productions must be in the form **A → γ**

where **A ∈ N** (Non terminal)

and **γ ∈ (T ∪ N)\*** (String of terminals and non-terminals).

These languages generated by these grammars are be recognized by a non-deterministic pushdown automaton.

Example

**S → Xa**

**X → a**

**X → aX**

**X → abc**

**X → ε**

Type - 1 Grammar

**Type-1 grammars** generate context-sensitive languages. The productions must be in the form

**α A β → α γ β**

where **A ∈ N** (Non-terminal)

and **α, β, γ ∈ (T ∪ N)\*** (Strings of terminals and non-terminals)

The strings **α** and **β** may be empty, but **γ** must be non-empty.

The rule **S → ε** is allowed if S does not appear on the right side of any rule. The languages generated by these grammars are recognized by a linear bounded automaton.

Example

**AB → AbBc**

**A → bcA**

**B → b**

Type - 0 Grammar

**Type-0 grammars**  generate recursively enumerable languages. The productions have no restrictions. They are any phase structure grammar including all formal grammars.

They generate the languages that are recognized by a Turing machine.

The productions can be in the form of **α → β** where **α** is a string of terminals and nonterminals with at least one non-terminal and **α** cannot be null. **β** is a string of terminals and non-terminals.

Example

**S → ACaB**

**Bc → acB**

**CB → DB**

**aD → Db**

1. A grammar for simple sentences. Things in this language are:
2. **THE MAN BITES A DOG**
3. **A DOG PETS A DOG**

Things not in this language are:

**MAN BITES DOG**

Here is the grammar:

**<sentence> ::= <subject> <predicate>**

**<subject> ::= <article> <noun>**

**<predicate> ::= <verb> <direct-object>**

**<direct-object> ::= <article> <noun>**

**<article> ::= THE | A**

**<noun> ::= MAN | DOG**

**<verb> ::= BITES | PETS**

1. An ambiguous grammar for binary numbers:
2. **<binary-string> ::= 0 | 1 | <binary-string> <binary-string>**
3. An unambiguous grammar for binary numbers:
4. **<binary-string> ::= <binary-string> 0 | <binary-string> 1 | 0 | 1**
5. An ambiguous grammar for some arithmatic expressions using + and - and variables X, Y and Z. This grammar also does not assert any precedence between + and - or associativity.
6. **<sentence> ::= <expression>**
7. **<expression> ::= <expression> + <expression> |**
8. **<expression> \* <expression> |**
9. **<identifier>**
10. **<identifier> ::= X | Y | Z**
11. An unambiguous grammar for some arithmatic expressions using + and - and variables X, Y and Z. THis asserts \* is performed before + and both \* and + are done left to right.
12. **<sentence> ::= <expression>**
13. **<expression> ::= <term> | <expression> + <term>**
14. **<term> ::= <identifier> | <term> \* <identifier>**
15. **<identifier> ::= X | Y | Z**
16. The following shows the ^ operator performed right to left
17. **<sentence> ::= <expression>**
18. **<expression> ::= <term> | <term> ^ <expression>**
19. **<term> ::= <identifier> | <term> \* <identifier>**
20. **<identifier> ::= X | Y | Z**
21. This is ambiguous. Why?
22. **<sentence> ::= <expression>**
23. **<expression> ::= <term> | <expression> + <term>**
24. **<term> ::= <identifier> | <term> \* <identifier> | <expression>**
25. **<identifier> ::= X | Y | Z**
26. This solves the ambiguity but doesn't handle all cases that you might want.
27. **<sentence> ::= <expression>**
28. **<expression> ::= <term> | <expression> + <term>**
29. **<term> ::= <identifier> | <term> \* <identifier> | ( <expression> )**
30. **<identifier> ::= X | Y | Z**
31. How is this language defined by this grammar different than the language above?
32. **<sentence> ::= <expression>**
33. **<expression> ::= <term> | <expression> + <term>**
34. **<term> ::= <identifier> | <term> \* <identifier>**
35. **<identifier> ::= X | Y | Z | ( <expression> )**
36. This is a grammar for a language that has signed digits.
37. **<signed> ::= + <nosign> | - <nosign>**
38. **<nosign> ::= <digit> <nosign> | <digit>**
39. **<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**
40. This is a grammar for an even number of X's
41. **<even> ::= <even> XX | XX**
42. This is a grammar for numbers with no leading zeros and signs allowed on nonzero numbers.
43. **<number> ::= + <nosign> | - <nosign> | <nosign> | 0**
44. **<nosign> ::= <nonzero> <anydig> | <nonzero>**
45. **<anydig> ::= 0 | <nozero>**
46. **<nozero> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**

Some more grammars with errors:

1. This is ambiguous
2. **<zot> ::= <zot> b <zot> | a**
3. This is ambiguous
4. **<zot> ::= <zing> | a**
5. **<zing> ::= <zot> | b**
6. This has a different sort of problem
7. **<zot> ::= <zing> | a**

**<zing> ::= <zing> b**

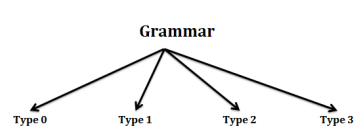
Chomsky Hierarchy is a broad classification of the various types of grammar available

These include Unrestricted grammar, context-free grammar, context-sensitive grammar and restricted grammar

Grammars are classified by the form of their productions.

Each category represents a class of languages that can be recognized by a different automaton

The classes are nested, with type 0 being the largest and most general, and type 3 being the smallest and most restricted.



**Type 0 :Unrestricted grammar :** This type of grammar requires LHS to contain atleast one non-terminal

**Applications:** Unrestricted language recursively enumerable automaton Turing machine.

– x → y – x ∈ ((V ∪ T)∗V (V ∪ T)∗) – y ∈ (V ∪ T)∗

– LHS contains at least one non-terminal

– generate recursively enumerable languages

– recognized by Turing Machines

– example :

generates L = {ww | w ∈ (a + b)\*}

Generate wCwR E

C is a temporary center marker

E is a wall

∗ Reverse wRwR by pushing symbols in wRwR, leftmost first, over the wall with a pusher P

∗ Clean up by removing C and E

V = {S, T, C, P}

T = {a, b} P :

1. S → TE
2. T → aTa
3. T→ | bTb
4. T→ | C
5. C → CP
6. Paa → aPa
7. Pab → bPa
8. Pba → aPb
9. Pbb → bPb
10. PaE → Ea
11. PbE → Eb
12. CE → є

**Type 1 : Context-sensitive grammar**

**Applications**: context-sensitive language context-sensitive automaton linear-bounded automaton – x → y

or S → є and S is not in RHS.

– x∈ ((V ∪ T)∗V (V ∪ T)∗)

– y∈ (V ∪ T)+ – |x| ≤ |y| – non-contracting

– generate context-sensitive languages

– recognized by linear-bounded automata

– example

V = {S, A, B}

T = {a, b, c}

P :

1. S → abc
2. | aAbc
3. Ab → bA
4. Ac → Bbcc
5. bB → Bb
6. aB → aa
7. aB → aaA

generates L =anbncn|n≥1anbncn|n≥1

**Type 2 : Context-free** -It is a grammar that is used to generate languages recursively and requires it to be in Chomsky Normal Form or the Griebach Normal Form

**Applications:** context-free language context-free automaton pushdown automaton

– x → y

– x∈ V

– y∈ (V ∪ T)\*

– generate context-free languages

– recognized by pushdown automata

– example

V = {S}

T = {a, b}

P :

1. S → aSb

2.S → |є

generates L = {a^nb^n| n ≥ 0}

**Type 3 : Regular grammar**

– A regular grammar is right-linear or left- linear (but not both)

– A, B ∈ V

– y∈ T\*

– Right-Linear productions have the form A → yB or A → y

– Left-Linear productions have the form A → By or A → y

– generate regular languages

– recognized by finite state automata

– example

V = {S}

T = {a}

P :

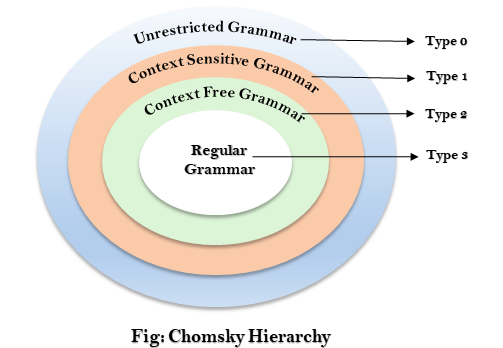
1. S → aS
2. . S → |є

generates L = an;|n≥0an;|n≥0

# Chomsky Hierarchy

Chomsky Hierarchy represents the class of languages that are accepted by the different machine. The category of language in Chomsky's Hierarchy is as given below:

1. Type 0 known as Unrestricted Grammar.
2. Type 1 known as Context Sensitive Grammar.
3. Type 2 known as Context Free Grammar.
4. Type 3 Regular Grammar.



This is a hierarchy. Therefore every language of type 3 is also of type 2, 1 and 0. Similarly, every language of type 2 is also of type 1 and type 0, etc.

### Type 0 Grammar:

Type 0 grammar is known as Unrestricted grammar. There is no restriction on the grammar rules of these types of languages. These languages can be efficiently modeled by Turing machines.

**For example:**

1. bAa → aa
2. S → s

### Type 1 Grammar:

Type 1 grammar is known as Context Sensitive Grammar. The context sensitive grammar is used to represent context sensitive language. The context sensitive grammar follows the following rules:

* The context sensitive grammar may have more than one symbol on the left hand side of their production rules.
* The number of symbols on the left-hand side must not exceed the number of symbols on the right-hand side.
* The rule of the form A → ε is not allowed unless A is a start symbol. It does not occur on the right-hand side of any rule.
* The Type 1 grammar should be Type 0. In type 1, Production is in the form of V → T

Where the count of symbol in V is less than or equal to T.

**For example:**

1. S → AT
2. T → xy
3. A → a

### Type 2 Grammar:

Type 2 Grammar is known as Context Free Grammar. Context free languages are the languages which can be represented by the context free grammar (CFG). Type 2 should be type 1. The production rule is of the form

1. A → α

Where A is any single non-terminal and is any combination of terminals and non-terminals.

**For example:**

1. A → aBb
2. A → b
3. B → a

### Type 3 Grammar:

Type 3 Grammar is known as Regular Grammar. Regular languages are those languages which can be described using regular expressions. These languages can be modeled by NFA or DFA.

Type 3 is most restricted form of grammar. The Type 3 grammar should be Type 2 and Type 1. Type 3 should be in the form of

1. V → T\*V / T\*

**For example:**

1. A → xy