Chomsky Hierarchy
Language Operations and Properties

Topic of Discussion:

- Introduction.
- Chomsky Hierarchy Of Languages.
- Types Of Languages:
  - Type - 0
  - Type - 1
  - Type - 2
  - Type - 3

Introduction:

- **Noam Chomsky,** is an **American linguist**, **philosopher**, **scientist** and **social activist.**
- Chomsky hierarchy of grammars was described by **Noam Chomsky** in **1956**.
- **Grammar Definition**: It is defined by four tuples: **G = {V,T,P,S}** where
  - V = Non Terminals
  - T = Terminals
  - P = Production Rule
  - S = Start Symbol

Production Rule:

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow b$

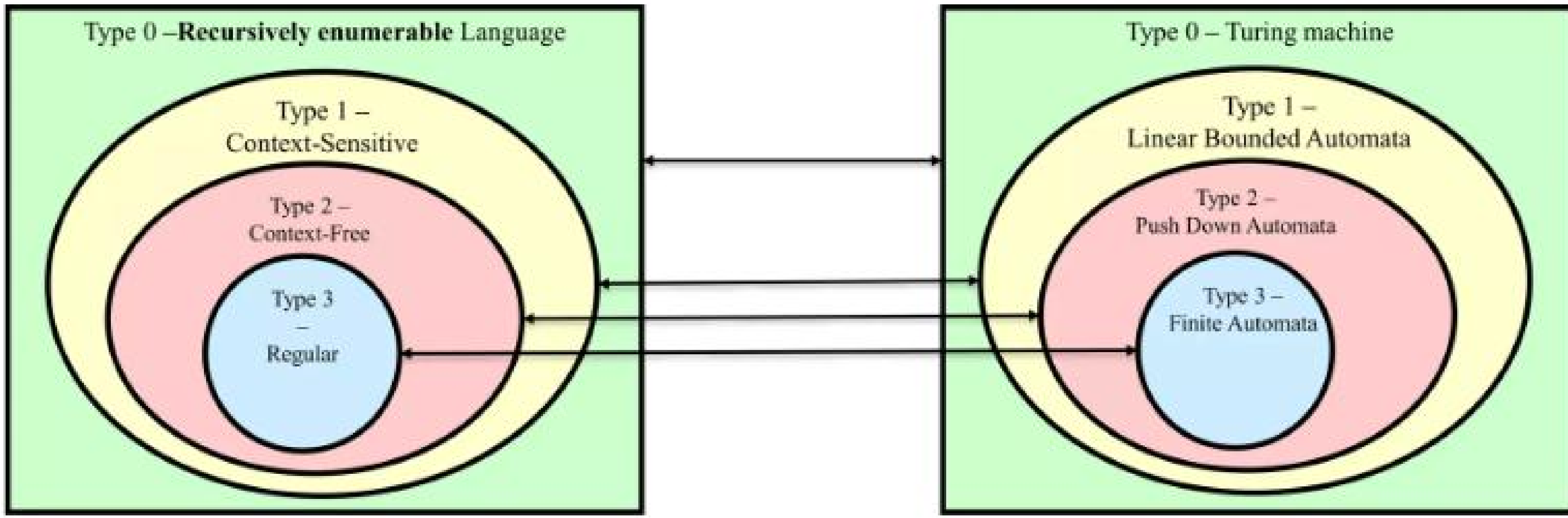
The Chomsky Hierarchy

Type	Language	Grammar	Automaton
0	Recursively Enumerable	Unrestricted	DTM - NTM
1	Context Sensitive	Context Sensitive	Linearly Bounded Automaton
2	Context Free	Context Free	NPDA
3	Regular	Right Linear, Left Linear	DFA, NFA

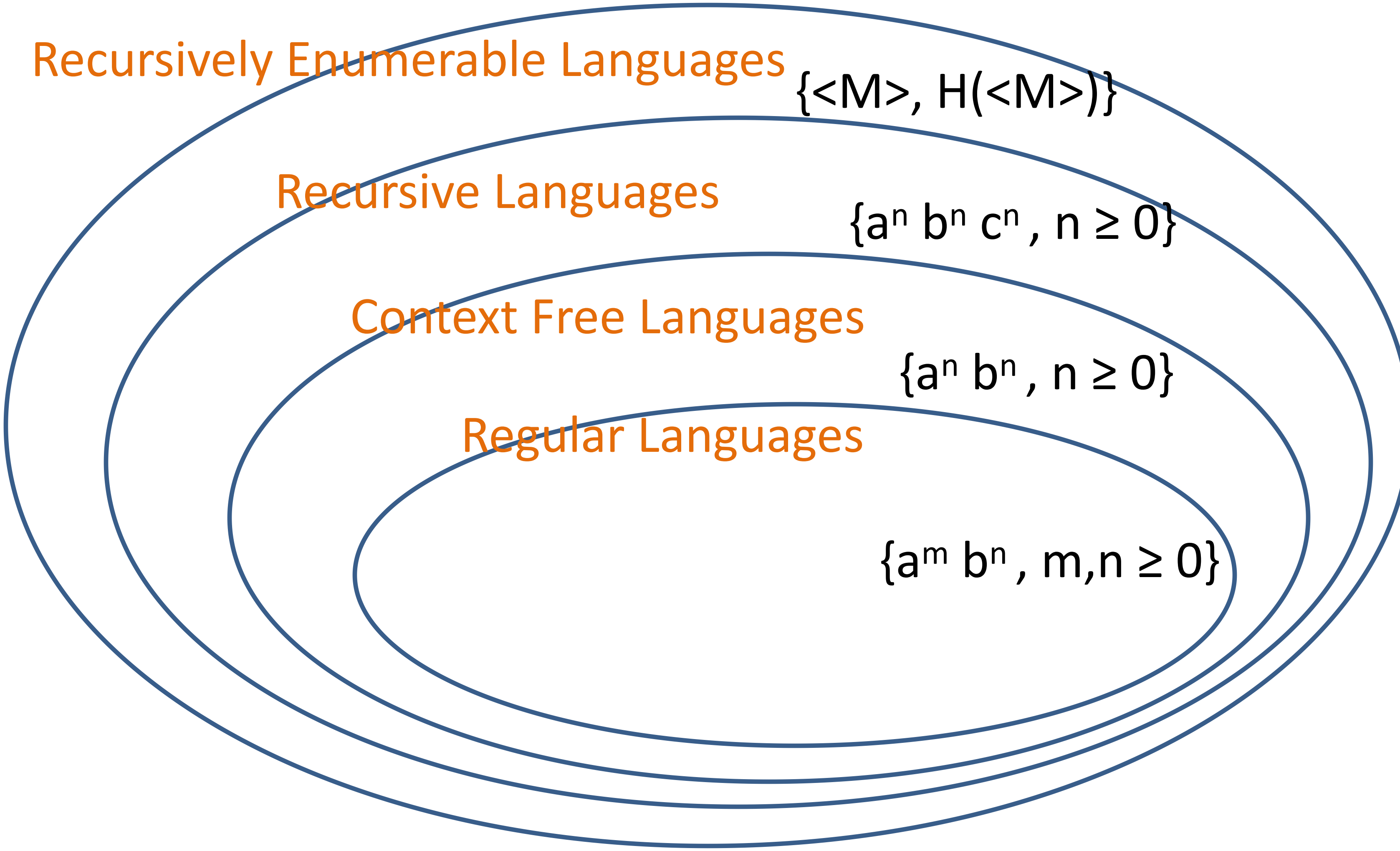
$$\text{Type3} \subset \text{Type2} \subset \text{Type1} \subset \text{Type0}$$

Chomsky Hierarchy Of Languages:

Venn Diagram of Grammar Types:



The Chomsky Hierarchy



Types Of Languages:

- *Recursively enumerable* Language *(Type-0)*
- *Context-sensitive* Language *(Type-1)*
- *Context-free* Language *(Type-2)*
- *Regular* Language *(Type-3)*

Type-0:

- **Type-0 Languages** (unrestricted grammars) include all formal grammars.
- They generate exactly all languages that can be recognized by a **Turing machine.**
- These languages are also known as the **recursively enumerable languages**.
- **Type-0 grammars** are too general to describe the syntax of programming languages and natural languages.
- This grammar has rules of the form $\alpha \rightarrow \beta$ (where a contains non terminal and β contains terminals or non terminals).
- Example:
- $AB \rightarrow A$ ✔
- $AB \rightarrow aB$ ✔
- $S \rightarrow$ ^ ✔
- $a \rightarrow AB$ ✖
- ^ $\rightarrow a$ ✖

α = alpha β = Beta

Type-1:

- **Type-1 grammar** generate the **context-sensitive languages**.
- The languages described by these grammars are exactly all languages that can be recognized by a **linear bounded automaton**.
- These grammars have rules of the form $\alpha \rightarrow \beta$ with **a restriction that length of** $|a| \leq |\beta|$.
- Example:
  - $aAb \rightarrow bbb$ ✔
  - $aA \rightarrow bbb$ ✔
  - $aAb \rightarrow bb$ ✖

Type-2:

- **Type-2 Languages** generate the **context-free languages**.
- These languages are exactly all languages that can be recognized by a **non-deterministic pushdown automaton**.
- Context-free languages are the theoretical basis for the syntax of most programming languages.
- These are defined by rules of the form $A \rightarrow \alpha$ where A is a nonterminal and α is string of terminals and nonterminal(there will be no context on the left and right of nonterminal).
- Example:
  - $A \rightarrow BCD$ ✔
  - $A \rightarrow aBC$ ✔
  - $a \rightarrow AbC$ ✖

Type-3:

- **Type-3 Languages** generate the regular languages.
- These languages are exactly all languages that can be decided by a finite state automaton
- Regular languages are commonly used to define search patterns of programming languages.
- It can be classified into two types **(1)Right Linear (2)Left Linear.**
- **If we have repetition of non terminals on right side[** $A \rightarrow xB|x$**] then it is known as Right Linear.**
- **If we have repetition of non terminals on left side[** $A \rightarrow Bx|x$**] then it is known as Left Linear.(A,B** $\in$ non terminals and $x \in \Sigma^*$)
- Example:
- $S \rightarrow aS|b$
- $S \rightarrow aS|c$
- $S \rightarrow Sa|b$
- $A \rightarrow ba$

Remarks

- **Type-1 grammars** are non-contracting (length increaing grammars) , meaning the length of the right-hand side must not be shorter than the left-hand side.
  - Hence, $A \rightarrow \varepsilon$ is disallowed because it reduces string length.
- **Type-2 grammars** (Context-Free) permit $A \rightarrow \varepsilon$ for simplification and flexibility in derivations.
  - Such rules help generate optional or empty structures.
- After applying the Null Production Removal Algorithm, ε-productions are systematically removed without changing the language.
  - The resulting grammar becomes non-contracting and thus fits Type-1 constraints.

Null Production

“after removing null productions, grammar becomes acceptable from Type 2 onward (i.e., for Type 1 too).”