A Hierarchy of Formal Languages and Automata

- How large is the family of languages accepted by Turing machines?

- Is there any language that cannot be accepted by Turing machines?
Recursively Enumerable Languages

A language $L$ is said to be recursively enumerable if there exists a Turing machine that accepts it:

$$q_0w \xrightarrow{*} M x_1q_fx_2 \quad q_f \in F$$

for every $w \in L$. 
Recursively Enumerable Languages

A language L is said to be recursively enumerable if there exists a Turing machine that accepts it:

\[ q_0 \ x \rightarrow^* M \ x_1 q_f x_2 \quad q_f \in F \]

for every \( w \in L \).

The definition says nothing about the case \( w \notin L \), regarding if M halts on w or not.
Recursive Languages

A language $L$ on $\Sigma$ is said to be **recursive** if there exists a Turing machine that accepts $L$ and that **halts** on every $w \in \Sigma^+$. 
Recursively Languages

There exists an easily constructed \textit{enumeration procedure} for a recursive language:

Enumerate every string and check if it is accepted by the TM of the given language.
Recursively Languages

An enumeration procedure for a recursively enumerable language:

Alternatively enumerate and test every string if it is accepted by the TM of the given language.
Theorem

There exist languages that are not recursively enumerable.

There exist languages that cannot be accepted by Turing machines.
Theorem

There exist languages that are not recursively enumerable.

Proof:
If $S$ is an infinite countable set, then $2^S$ is not countable.
⇒ the set of all languages $2^{\Sigma^*}$ is not countable.

The set of all TMs is countable, so the set of all recursively enumerable languages is also countable.
⇒ theorem proved.
Theorem

There exist recursively enumerable languages that are not recursive.
Unrestricted Grammars

G = (V, T, S, P)

Productions are of the form:

\[ u \rightarrow v \]

\[ u \in (V \cup T)^+ \text{ and } v \in (V \cup T)^* \]
Theorem

Any language generated by an unrestricted grammar is recursively enumerable.
Theorem

For every recursively enumerable language, there exists an unrestricted grammar that generates it.
Context-Sensitive Grammars

\[ G = (V, T, S, P) \]

Productions are of the form:

\[ x \rightarrow y \]

\[ x, y \in (V \cup T)^+ \text{ and } |x| \leq |y| \]
Example

- \( L = \{a^n b^n c^n \mid n \geq 1\} \) is context-sensitive.

\[
G = (\{S, A, B\}, \{a, b\}, S, P) \\
P = \{ S \rightarrow abc \mid aAbc \\
\hspace{1cm} Ab \rightarrow bA \\
\hspace{1cm} Ac \rightarrow Bb \hspace{1cm} Bccc \\
\hspace{1cm} bB \rightarrow Bb \\
\hspace{1cm} aB \rightarrow aa \mid aaA \\
\hspace{1cm} aB \rightarrow aa \mid aaA \}
\]
Linear Bounded Automata

Control unit

\[ q_0 \]
Theorem

For every context-sensitive $L$ not containing $\lambda$, there exists a LBA $M$ such that $L = L(M)$. 
Theorem

If a language is accepted by some LBA $M$, then there exists a context-sensitive grammar $G$ such that $L(G) = L(M)$.
Context-Sensitive and Recursive Languages

- Every context-sensitive language is recursive.
- There exists recursive languages that are not context-sensitive.
The Chomsky Hierarchy

- $L_{REG}$: type 3 (FA)
- $L_{CF}$: type 2 (NPDA)
- $L_{CS}$: type 1 (LBA)
- $L_{REN}$: type 0 (TM)
Decidability

- A **decision problem** is a problem that asks a question that has a **yes** or **no** answer.
Decidability

- The **Satisfiability Problem**: Is a logical formula satisfiable?
Decidability

- The **Halting Problem**: Is a program halts on a certain input?
Decidability

• A decision problem is **solvable** (decidable) if there is an algorithm that **halts with the correct answer** for any instance of the problem.

• If no such algorithm exists, then the problem is **unsolvable** (undecidable).
Decidability

• A decision problem is partially solvable (semi-decidable) if there is an algorithm that halts with the answer yes for those instances of the problem whose answers are yes, but may run forever for those instances of the problem whose answers are no.
Decidability

• The Satisfiability Problem is **decidable** for Propositional Logic.

• The Satisfiability Problem is **semi-decidable** for Predicate Logic.
The Halting Problem

- Is there an algorithm that can decide whether the execution of an arbitrary program halts on an arbitrary input?
The Halting Problem

• Is there a Turing machine that can decide whether the execution of an arbitrary Turing machine halts on an arbitrary input string?
Theorem

• The Halting Problem is \textit{undecidable}. 
Theorem

• The Halting Problem is **undecidable**.

**Proof:**

If the Halting Problem were decidable, then every recursively enumerable language would be recursive.
Undecidable Problems for Recursively Enumerable Languages

Let $G$ be an unrestricted grammar.

The problem of determining whether or not $L(G) = \emptyset$ is undecidable.
Undecidable Problems for Recursively Enumerable Languages

Let $M$ be a Turing machine.

The problem of determining whether or not $L(M)$ is finite is undecidable.
Undecidable Problems for Context-Free Languages

There exist no algorithm for deciding whether or not any given context-free grammar is ambiguous.
Undecidable Problems for Context-Free Languages

There exist no algorithm for deciding whether or not two arbitrary context-free languages are disjoint.