**Definition:** Recursively Enumerable Language (REL): a language \( L \) on \( \Sigma \) is recursively enumerable if there exists a Turing machine \( T_m \) that accepts \( L \) and halts on every \( w \in \Sigma^+ \). (i.e., there exists a membership algorithm for \( L \).

**Theorem 1.3:** The set of all Turing machines \( \mathcal{T}_m \) is infinite but countable.

(\( \mathcal{T}_m \) is infinite because each description contains \( \star \) for \( 0, 1, \cdots \).

Each Turing description is finite and unique: \( \mathcal{T}_m \) can enumerate \( 00, 01, 10, \cdots \) to find which one represents \( T_m \).

**Proposition 1.1:** \( 2^\Sigma \) is not a countable set for \( \Sigma \) denumerable.

\[
S = \{ s_1, s_2, \ldots, s_n \}, \quad \exists \, t \in 2^\Sigma \text{ then } t = \{ 0, 100100 \ldots \}
\]

Let \( t = \{ 9, 5, 6 \} \).

Use diagonalization:

- \( t = \{ 0, 10, 10 \} \)
- \( t = \{ 0, 11, 10 \} \)
- \( t = \{ 0, 1, 10 \} \)

... not in \( 2^\Sigma \).

**Theorem 2:** For any non-empty \( \Sigma \), there exist languages that are not REL.

Set of all languages for \( \Sigma \): \( 2^{\Sigma^*} \) (uncountable). Whereas the set of \( \mathcal{T}_m \) is countable.

**Example:** Language not REL.

\( \exists \) a REL \( L \), s.t. \( L \notin \text{REL} \)

Let \( \Sigma = \{ a \} \)

Consider set \( \mathcal{S} \) of all Turing machines on alphabet \( \Sigma \).

Order them \( \mathcal{T} \): Turing countable.

Proof (by contradiction):

- \( \exists \) \( L \in \text{REL}, \exists \mathcal{M}_L \in \mathcal{T} \), s.t. \( L = \{ a \} \)

**Enumerative Procedure:**

```
for \( \mathcal{T} \in \mathcal{S} \):
    if \( \mathcal{T} \) is not REL:
        add \( \mathcal{T} \) to the list of REL languages.
```

- Generate all strings \( \{ a \}^* \)
  - \( e.g., \{ a, a^2, a^3 \} \)
  - \( w_1, w_2, \ldots \), \( w_1 = 0, w_2 = 1, 01, 10, \ldots \).

- For \( w \in \{ a \}^* \), \( w = w_1 w_2 \ldots \), \( w \in L \mathcal{T} \) if \( L \mathcal{T} \) is REL.

- Any \( w \) will be caught.
Theorem:

If \( L \in \text{REL} \) and \( T \in \text{REL} \), then \( L \leq \text{Rel} \) and \( T \leq \text{Rel} \) also.

Proof:

If \( L \in \text{REL} \) and \( T \in \text{REL} \),

Find \( \hat{T} \) for \( L \) and \( \hat{T} \) for \( T \).

The products \( \hat{T}_1, \hat{T}_2, \ldots \) + \( \hat{T}_1 \) produce \( \hat{T}_1, \hat{T}_2, \ldots \)

every \( w \in L \) or \( w \in T \).

Given \( w \), we continue with \( w, \hat{T}_1, \hat{T}_2, \ldots, \hat{T}_n \),

and we will end up string.

Thus we can be found \( w \in L \) or \( T \).

This is the membership algorithm for \( L \) and \( T \).

\[ L = \{ a^i : a^i \in L \text{ and } i \leq \infty \} \in \text{REL} \text{ but } T \notin \text{REL} \]

\[ L \leq \text{REL} \text{ but } L \notin \text{Rel} \]
Unrestricted Grammar

Defn: \( G = (V, \Sigma, S, P) \) if \( P: \text{u} \rightarrow \text{v}, \text{u} \in (V \cup T)^+ \) and \( \text{v} \in (V \cup T)^* \)

Thus, \( G(\text{REL}) = \text{unrestricted grammar} \)

& \( \text{check } L(G) = \text{REL} \)

1. \( L(UG) = \text{REL} \)
   - Any grammar has finite production rules.
     - Collect strings \( S \rightarrow \text{w} \)
     - that have one step derivation, then
       - \( 2 \text{ steps etc.} \)
     - \( S \rightarrow S \rightarrow \text{w} \)

\( \text{w} \) are enumerable.

Converse:
- Given a TM produce REL \( G \) s.t. \( L(TM) = L(G) \).

Constructive proof for creating \( G \).
- For computation of a TM
  - \( \delta_0(w) \xrightarrow{\star} xq_y \)  \( \forall q \in \Sigma \)
  - Produce grammar \( G_G \)
    - \( \delta_0(w) \rightarrow xq \)

  We need: \( S \xrightarrow{\star} \delta_0 \rightarrow xq_y \rightarrow w \)

Context Sensitive Grammars (CSG)

Defn: \( G = (V_T, \Sigma, \text{S}, P) \) of CSG if \( P: \text{x} \rightarrow \text{y} \), \( \text{x} \in \Sigma \text{G} \), \( \text{y} \in (V_T)^+ \)

and \( |x| \leq |y| \)

(Thus, CFGs can be written in normal form:
  \( xAy \rightarrow zv \text{y} \) \( \forall \text{A} \rightarrow \text{v} \) only if \( \text{x} \) is context of \( \text{z} \) and \( \text{y} \))
Content: Sensitive Language:
CSL = \{ L(CSL) \mid L(CSL) \cup \{ \lambda \} \}

For every CSL \( \{ \} \cup \{ \} \), \( \exists \) a LBA (Linear Bounded Automata) s.t. CSL = L(LBA).

Compare to Greibach normal form for CFL or Chomsky for CFL, we can see that CFL \( \subseteq \) CSL.

\( L = \{ a^m b^n c^n \mid m \geq n > 0 \} \) is a CSL and not CFL.

\( \exists \) LBA for every CSL \( \Rightarrow \) every CSL is REL.

\( \exists \) LBA for every CSL \( \Rightarrow \) every CSL is REL.

Moreover, CSL are recursive \( \Leftrightarrow \) REL.

\( \Rightarrow \) They have a \( \sum \) CSG and it can be proved
that \( \exists \) a membership for it.

\( \Rightarrow \) For a REL that is not CSL.

\( \Rightarrow \) For a REL that is not CSL.

We can have a coding for all CSG (i.e. also languages).

Define the strings \{0,1\}^* \( \cup \{ 0, w_1, w_2 \ldots \} \) (descriptions). Order the strings \{0,1\}^* \( \cup \{ 0, w_1, w_2 \ldots \} \).

By definition, \( w_j \) may not define a CSL, if it does call the grammar \( G_j \).

Define \( L = \{ w_i \mid w_i \) defines a CSL \( G_i \) and \( w_i \notin L(G_i) \} \).

We denote \( L \subseteq \) REL.

But, \( L \notin \) CSL.

\( \Rightarrow \) If \( L \), there would exist some \( w_j \) s.t. \( L = L(G_j) \).

\( \rightarrow \) Contradiction.

\( \Rightarrow \) If \( w_i \notin L(G_i) \) then \( w_j \notin L \), but \( L \). \( \subseteq \) L(G) \( \Leftrightarrow \).

\( \Rightarrow \) If \( w_j \notin L(G_j) \) then \( w_j \notin L \).

\( \Rightarrow \) L \( \notin \) CSL.
Chomsky Hierarchy:

- REL (type 0)
- CSL (type 1)
- CFL (type 2)
- REG (type 3)

Other hierarchies possible.