The Chomsky Hierarchy

We can define four types of grammars by restricting the form of productions $\alpha \rightarrow \beta$ for $\alpha, \beta \in (V \cup T)^*$.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Restriction</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>RE</td>
<td>None</td>
<td>TM</td>
</tr>
<tr>
<td>1</td>
<td>CSL</td>
<td>$</td>
<td>\alpha</td>
</tr>
<tr>
<td>2</td>
<td>CFL</td>
<td>$\alpha \in V$</td>
<td>PDA</td>
</tr>
<tr>
<td>3</td>
<td>Regular</td>
<td>$\alpha \in V$, $\beta \in T^<em>V \cup T^</em>$</td>
<td>DFA</td>
</tr>
</tbody>
</table>

Computational Complexity

**Defn:** Let $T : N \rightarrow N$. A TM $M$ is said to have *time complexity* $T(n)$ if on every input string $w$, $M$ takes no more than $T(|w|)$ transitions.

**Defn:** $\mathcal{P}$ is the set of all languages $L \subseteq \{0,1\}^*$ such that there is a polynomial $p(n)$ and a TM $M$ with time complexity $p(n)$ such that $L(M) = L$.

**Defn:** $\mathcal{NP}$ is the set of all languages $L \subseteq \{0,1\}^*$ such that there is a polynomial $p(n)$ and a nondeterministic TM $M$ with time complexity $p(n)$ such that $L(M) = L$. 
Motivation

- \( \mathcal{P} \) represents the set of decision problems that can be decided by polynomial-time algorithms (see Section 8.6).
- The set of 1-variable polynomials over \( \mathbb{N} \) is closed under addition, multiplication, and composition.
- If there is a polynomial-time algorithm for a problem, there is usually an efficient one.
- There are many interesting problems in \( \mathcal{NP} \) that are not known to be in \( \mathcal{P} \).

Claim: \( \mathcal{P} \subseteq \mathcal{NP} \).

Open Question: Is \( \mathcal{P} = \mathcal{NP} \)?

Working Hypothesis: \( \mathcal{P} \neq \mathcal{NP} \).

Defn: A language \( L_1 \) is polynomially many-one reducible to a language \( L_2 \) (\( L_1 \leq_p L_2 \)) if there is a TM with polynomial time complexity that reduces \( L_1 \) to \( L_2 \).

Claim: If \( L_1 \leq_p L_2 \), \( L_1 \subseteq \{0,1\}^* \), and \( L_2 \in \mathcal{P} \), then \( L_1 \in \mathcal{P} \).

Defn: A language \( L \) is said to be \( \mathcal{NP} \)-hard if for every \( L' \in \mathcal{NP} \), \( L' \leq_p L \). If, in addition, \( L \in \mathcal{NP} \), then \( L \) is said to be \( \mathcal{NP} \)-complete.
Theorem 10.5: If $L \in \mathcal{P}$ is $\mathcal{NP}$-complete, then $\mathcal{P} = \mathcal{NP}$.

Proof: Let $L' \in \mathcal{NP}$.

- Because $L$ is $\mathcal{NP}$-complete, $L' \leq^p_m L$.
- Because $L \in \mathcal{P}$, $L' \in \mathcal{P}$.
- Because $\mathcal{P} \subseteq \mathcal{NP}$, it follows that $\mathcal{P} = \mathcal{NP}$.

We therefore consider $\mathcal{NP}$-completeness of a language $L$ to be strong evidence that $L \not\in \mathcal{P}$.

Claim: If $L_1 \leq^p_m L_2$ and $L_2 \leq^p_m L_3$, then $L_1 \leq^p_m L_3$.

Corollary: If $L_1$ is $\mathcal{NP}$-hard and $L_1 \leq^p_m L_2$, then $L_2$ is $\mathcal{NP}$-hard.

We can therefore show a language to be $\mathcal{NP}$-hard by reducing a known $\mathcal{NP}$-hard language to it.

In order to get our first $\mathcal{NP}$-hard language, we must reduce every language in $\mathcal{NP}$ to it.
Boolean Satisfiability (SAT)

**Input:** A boolean formula $\mathcal{F}$ consisting of boolean variables and the operators $\land$, $\lor$, and $\neg$.

**Question:** Is there an assignment of boolean values to the variables in $\mathcal{F}$ that causes $\mathcal{F}$ to evaluate to $\text{true}$?

**Claim:** $L_{\text{SAT}} \in \mathcal{NP}$, where $L_{\text{SAT}}$ denotes the language of satisfiable formulas encoded over $\{0, 1\}$.

---

**Cook’s Theorem:** $\text{SAT}$ is $\mathcal{NP}$-hard.

**Proof idea:**

- For each language $L$ in $\mathcal{NP}$, there is a polynomial $p(n)$ and a nondeterministic TM $M$ with time complexity $p(n)$ such that $L(M) = L$.

- From $w \in \{0, 1\}^*$, we construct a formula $\mathcal{F}$ that is satisfiable iff there is an accepting computation of $M$ on $w$.

- The time for the construction will be polynomial in $p(n)$. 
Construction overview:

- We will view a computation as a sequence of IDs \( \alpha_0, \ldots, \alpha_{p(n)} \) such that either \( \alpha_i \vdash \alpha_{i+1} \) or \( \alpha_i = \alpha_{i+1} \).
- Each \( \alpha_i \) will be of the form \( X_{-p(n)} \cdots X_0 \cdots X_{p(n)+1} \) where \( X_j \) is either a tape symbol or a state.
- We use boolean variable \( y_{ijA} \) to denote whether symbol \( X_j \) of \( \alpha_i \) is \( A \).
- \( \mathcal{F} \) will constrain the sequence of IDs to be an accepting computation on \( w \).

We will describe a set of formulas, each enforcing certain constraints on the variables \( y_{ijA} \), for \( 0 \leq i \leq p(n), -p(n) \leq j \leq p(n) + 1, A \in Q \cup \Gamma. \) \( \mathcal{F} \) will be the conjunction of these formulas.

\( \alpha_0 \) is the initial ID:

- \( y_{00q_0} \)
- \( y_{0ja_j} \) for \( 1 \leq j \leq n \), where \( a_1 \cdots a_n = w \).
- \( y_{0jB} \) for \( -p(n) \leq j < 0, n < j \leq p(n) + 1 \).

\( \alpha_{p(n)} \) contains a final state:

\[
\bigvee_{j=-p(n)}^{p(n)+1} \bigvee_{q \in F} y_{p(n)jq}
\]
- We still need to enforce that $\alpha_i \models \alpha_{i+1}$ or $\alpha_i = \alpha_{i+1}$ for $0 \leq i < p(n)$.
- For $0 \leq i < p(n)$, $-p(n) \leq j \leq p(n) + 1$, we construct a formula enforcing one of the following
  1. $X_{ij}$ is a state and $X_{i+1,j-1}X_{i+1,j}X_{i+1,j+1}$ results from doing nothing or taking a transition of $M$ from $X_{i,j-1}X_{ij}X_{i,j+1}$ (if $j = -p(n)$ or $j = p(n) + 1$, this is omitted); or
  2. $X_{i,j-1}$, $X_{ij}$, and $X_{i,j+1}$ are not states, and $X_{i+1,j} = X_{ij}$.

Constraint 1 is enforced by the disjunction of the following formulas:

- For each $q \in Q$, $X, Y \in \Gamma$, and $(q', Z, R) \in \delta(q, Y)$:
  $$y_{i,j-1,X} \land y_{i+1,j-1,X} \land y_{ijq} \land y_{i+1,j,Z} \land y_{i,j+1,Y} \land y_{i+1,j+1,q'}.$$
- For each $q \in Q$, $X, Y \in \Gamma$, and $(q', Z, L) \in \delta(q, y)$:
  $$y_{i,j-1,X} \land y_{i+1,j-1,q'} \land y_{ijq} \land y_{i+1,j,X} \land y_{i,j+1,Y} \land y_{i+1,j+1,Z}.$$
- For each $q \in Q$, $X, Y \in \Gamma$: $y_{i,j-1,X} \land y_{i+1,j-1,X} \land y_{ijq} \land y_{i+1,j,q} \land y_{i,j+1,Y} \land y_{i+1,j+1,Y}$. 


Constraint 2 is enforced by the conjunction of:

- $\bigvee_{X \in \Gamma} y_{i,j-1}, x$;
- $\bigvee_{X \in \Gamma} (y_{ij} x \land y_{i+1,j}, x)$; and
- $\bigvee_{x \in \Gamma} y_{i,j+1}, x$.

Conjuncts containing out-of-bounds subscripts are omitted.

- The formula can be constructed in polynomial time.
- The formula is satisfiable iff $M$ has an accepting computation on $w$.
- Therefore, SAT is $\mathcal{NP}$-hard.