Computational Complexity

Defn: Let $T : \mathbb{N} \rightarrow \mathbb{N}$. 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 $\mathcal{P} = \mathcal{NP}$?

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

Claim: If $L_1 \leq^p_m L_2$, $L_1 \subseteq \{0, 1\}^*$, 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 $L' \in \mathcal{NP}$, $L' \leq^p_m L$. If $L \in \mathcal{NP}$, 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 $L \notin \mathcal{P}$.
Claim: If $L_1 \leq_m L_2$ and $L_2 \leq_m L_3$, then $L_1 \leq_m L_3$.

Corollary: If $L_1$ is \$NP\$-hard and $L_1 \leq_m L_2$, then $L_2$ is \$NP\$-hard.

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

In order to get our first \$NP\$-hard language, we must reduce every language in \$NP\$ to it.

---

**Boolean Satisfiability (SAT)**

**Input:** A boolean formula $\mathcal{F}$ consisting of boolean variable 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 $true$?

**Claim:** $L_{SAT} \in \mathcal{NP}$, where $L_{SAT}$ denotes the language of satisfiable formulas encoded over \{0, 1\}.
Cook’s Theorem: \( SAT \) 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 \( \alpha_i \downarrow \alpha_{i+1} \) 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 \( y_{ij}A \) 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 \( w \).