Lesson 0: Preliminaries

Theme: Review of some introductory material.

Let $\mathbb{N}$ denote the set of natural numbers $\{0, 1, 2, \ldots\}$. Let f and g be functions from $\mathbb{N}$ to $\mathbb{N}$.

- $f = O(g)$ means that there is c and n_0 such that for every $n \geqslant n_0$, $f(n) \leqslant c \cdot g(n)$.
  It is usually phrased as "there is c such that for (all) sufficiently large n," $f(n) \leqslant c \cdot g(n)$.
- $f = \Omega(g)$ means $g = O(f)$.
- $f = \Theta(g)$ means $g = O(f)$ and $f = O(g)$.
- $f = o(g)$ means for every $c > 0$, $f(n) \leqslant c \cdot g(n)$ for sufficiently large n.
  Equivalently, $f = o(g)$ means $f = O(g)$ and $g \neq O(f)$.
  Another equivalent definition is $f = o(g)$ means $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.
- $f = \omega(g)$ means $g = o(f)$.

To emphasize the input parameter, we will write $f(n) = O(g(n))$. The same for the Ω, o, ω notations. We also write $f(n) = \mathsf{poly}(n)$ to denote that $f(n) = c \cdot n^k$ for some c and $k \geqslant 1$.

Throughout the course, for an integer $n \geqslant 0$, we will denote by $\lfloor n \rfloor$ the binary representation of n. Likewise, $\lfloor G \rfloor$ the binary encoding of a graph G. In general, we write $\lfloor X \rfloor$ to denote the encoding/representation of an object X as a binary string, i.e., a 0-1 string. To avoid clutter, we often write X instead of $\lfloor X \rfloor$.

We usually use Σ to denote a finite input alphabet. Often $\Sigma = \{0, 1\}$. Recall also that for a word $w \in \Sigma^*$, $|w|$ denotes the length of w. For a DTM/NTM $\mathcal{M}$, we write $L(\mathcal{M})$ to denote the language $\{w : \mathcal{M} \text{ accepts } w\}$.

We often view a language $L \subseteq \Sigma^*$ as a boolean function, i.e., $L : \Sigma^* \to \{\mathsf{true}, \mathsf{false}\}$, where $L(x) = \mathsf{true}$ if and only if $x \in L$, for every $x \in \Sigma^*$.

1 Time complexity

Definition 0.1 Let $\mathcal{M}$ be a DTM/NTM, $w \in \Sigma^*$, $t \in \mathbb{N}$ and let $f : \mathbb{N} \to \mathbb{N}$ be a function.

- $\mathcal{M}$ *decides* w *in time* t *(or, in* t *steps)*, if every run of $\mathcal{M}$ on w has length at most t. That is, for every run of $\mathcal{M}$ on w:

  $$C_0 \vdash C_1 \vdash \cdots \vdash C_m \qquad \text{where } C_m \text{ is a halting configuration,}$$

  we have $m \leqslant t$.
- $\mathcal{M}$ *runs in time* $O(f(n))$, if there is $c > 0$ such that for sufficiently long word w, $\mathcal{M}$ decides w in time $c \cdot f(|w|)$.
- $\mathcal{M}$ *decides/accepts a language* L *in time* $O(f(n))$, if $L(\mathcal{M}) = L$ and $\mathcal{M}$ runs in time $O(f(n))$.
- $\textsc{Dtime}[f(n)] \stackrel{\mathsf{def}}{=} \{L : \text{there is a DTM } \mathcal{M} \text{ that decides } L \text{ in time } O(f(n))\}$.
- $\textsc{Ntime}[f(n)] \stackrel{\mathsf{def}}{=} \{L : \text{there is an NTM } \mathcal{M} \text{ that decides } L \text{ in time } O(f(n))\}$.

We say that $\mathcal{M}$ runs in *polynomial* and *exponential time*, if there is $f(n) = \mathsf{poly}(n)$ such that $\mathcal{M}$ runs in time $O(f(n))$ and $O(2^{f(n)})$, respectively. In this case we also say that $\mathcal{M}$ is a polynomial/exponential time TM.

The following are some of the important classes in complexity theory.

$$\mathbf{P} \stackrel{\mathsf{def}}{=} \bigcup_{f(n)=\mathsf{poly}(n)} \textsc{Dtime}[f(n)] \qquad \mathbf{EXP} \stackrel{\mathsf{def}}{=} \bigcup_{f(n)=\mathsf{poly}(n)} \textsc{Dtime}[2^{f(n)}]$$

$$\mathbf{NP} \stackrel{\mathsf{def}}{=} \bigcup_{f(n)=\mathsf{poly}(n)} \textsc{Ntime}[f(n)] \qquad \mathbf{NEXP} \stackrel{\mathsf{def}}{=} \bigcup_{f(n)=\mathsf{poly}(n)} \textsc{Ntime}[2^{f(n)}]$$

$$\mathbf{coNP} \stackrel{\mathsf{def}}{=} \{L : \Sigma^* - L \in \mathbf{NP}\} \qquad \mathbf{coNEXP} \stackrel{\mathsf{def}}{=} \{L : \Sigma^* - L \in \mathbf{NEXP}\}$$

Theorem 0.2 (Padding theorem) *If* $\mathbf{NP} = \mathbf{P}$*, then* $\mathbf{NEXP} = \mathbf{EXP}$.
Likewise, if $\mathbf{NP} = \mathbf{coNP}$*, then* $\mathbf{NEXP} = \mathbf{coNEXP}$.

2 Alternative definitions of the class NP

Note that according to the definition in the previous section, the class **NP** can be defined as follows.

Definition 0.3 A language L is in **NP** if there is $f(n) = \mathsf{poly}(n)$ and an NTM $\mathcal{M}$ such that $L(\mathcal{M}) = L$ and $\mathcal{M}$ runs in time $O(f(n))$.

There is an alternative definition of **NP**.

Definition 0.4 A language $L \subseteq \Sigma^*$ is in **NP** if there is a language $K \subseteq \Sigma^* \times \Sigma^*$ such that the following holds.

- For every $w \in \Sigma^*$, $w \in L$ if and only if there is $v \in \Sigma^*$ such that $(w, v) \in K$.
- There is $f(n) = \mathsf{poly}(n)$ such that for every $(w, v) \in K$, $|v| \leqslant f(|w|)$.
- The language K is accepted by a polynomial time DTM.

For $(w, v) \in K$, the string v is called the *certificate/witness* for w. We call the language K the *certificate/witness language* for L.

Indeed Def. 0.3 and 0.4 are equivalent. That is, for every language L, L is in **NP** in the sense of Def. 0.3 if and only if L is in **NP** in the sense of Def. 0.4.

3 NP-complete languages

Recall that a DTM $\mathcal{M}$ computes a function $F : \Sigma^* \to \Sigma^*$ in time $O(g(n))$, if there is a constant $c > 0$ such that on every word w, $\mathcal{M}$ computes $F(w)$ in time $\leqslant cg(|w|)$. If $g(n) = \mathsf{poly}(n)$, such fucntion F is called *polynomial time computable* function. Moreover, if $\mathcal{M}$ uses only logarithmic space, it is called *logarithmic space computable* function.

Definition 0.5 A language L_1 is *polynomial time reducible* to another language L_2, denoted by $L_1 \leqslant_p L_2$, if there is a polynomial time computable function F such that for every $w \in \Sigma^*$:

$$w \in L_1 \quad \text{if and only if} \quad F(w) \in L_2$$

Such function F is called polynomial time reduction, also known as *Karp reduction.*

If F is logarithmic space computable function, we say that L_1 is *log-space reducible* to L_2, denoted by $L_1 \leqslant_{\log} L_2$.

If L_1 and L_2 are in **NP** with certificate languages K_1 and K_2, respectively, we say that F is *parsimonious*, if for every $w \in \Sigma^*$, w has the same number of certificates in K_1 as $F(w)$ in K_2.

Definition 0.6 Let L be a language.

- L is **NP**-*hard*, if for every $L' \in$ **NP**, $L' \leqslant_p L$.
- L is **NP**-*complete*, if $L \in$ **NP** and L is **NP**-hard.

Recall that a propositional formula (Boolean expression) with variables $x_1, \ldots, x_n$ is in Conjunctive Normal Form (CNF), if it is of the form: $\bigwedge_i \bigvee_j \ell_{i,j}$ where each $\ell_{i,j}$ is a literal, i.e., a variable x_k or its negation $\neg x_k$. It is in 3-CNF, if it is of the form $\bigwedge_i \left(\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}\right)$. A formula φ is satisfiable, if there is an assignment of Boolean values true or false to each variable in φ that evaluates to true.

SAT	
Input:	A propositional formula φ in CNF.
Task:	Output true, if φ is satisfiable. Otherwise, output false.

3-SAT	
Input:	A propositional formula φ in 3-CNF.
Task:	Output true, if φ is satisfiable. Otherwise, output false.

Obviously, SAT can be viewed as a language, i.e., SAT $\stackrel{\text{def}}{=} \{\varphi : \varphi \text{ is satisfiable CNF formula}\}$. Likewise, for 3-SAT.

Theorem 0.7 (Cook 1971, Levin 1973) SAT *and* 3-SAT *are* **NP**-*complete.*

4 coNP-complete problems

Analogous to **NP**-complete, we can also define **coNP**-complete problems.

Definition 0.8 Let K be a language.

- K is **coNP**-*hard*, if for every $L \in$ **coNP**, $L \leqslant_p K$.
- K is **coNP**-*complete*, if $K \in$ **coNP** and K is **coNP**-hard.

Note that for every language K, K is **NP**-complete if and only if its complement $\overline{K}$ is **coNP**-complete, where $\overline{K} \stackrel{\text{def}}{=} \Sigma^* - K$. Thus, $\overline{\text{SAT}} \stackrel{\text{def}}{=} \{\varphi : \varphi \text{ is not satisfiable}\}$ is **coNP**-complete.