Lesson 3: More on the class NP

Theme: Some classical results on the class **NP**.

1 Ladner's theorem: NP-intermediate language

Theorem 3.1 (Ladner 1975) *If* $\mathbf{P} \neq \mathbf{NP}$*, then there is* $L \in \mathbf{NP}$ *such that* $L \notin \mathbf{P}$ *and* L *is not* $\mathbf{NP}$*-complete.*

For a function $f : \mathbb{N} \to \mathbb{N}$, we say that it is *polynomial time computable* (in unary representation), if there is a polynomial time algorithm that on input 1^n, outputs $1^{f(n)}$.

For a function $f : \mathbb{N} \to \mathbb{N}$, define SAT_f as follows.

$$\mathsf{SAT}_f \quad \stackrel{\text{def}}{=} \quad \{\varphi 0 \underbrace{1 \cdots 1}_{n^{f(n)}} : \varphi \in \mathsf{SAT} \text{ and } |\varphi| = n\}$$

We first prove the following lemma.

Lemma 3.2 *Suppose* $\mathbf{NP} \neq \mathbf{P}$*. If* $h : \mathbb{N} \to \mathbb{N}$ *is polynomial time computable (in unary representation), non-decreasing and unbounded, i.e.,* $\lim_{n\to\infty} h(n) = \infty$*, then* SAT_h *is not* $\mathbf{NP}$*-hard.*

Proof. Suppose to the contrary that SAT_h is **NP**-hard. Let F be a polynomial time reduction from SAT to SAT_h that runs in time cn^k. Let N be an integer such that for every $n \geqslant N$, the following holds.

- $h(n) \geqslant 2k$. (This is possible because h is non-decreasing and unbounded.)
- $cn^{1/2} < n$.

Claim 1 *For every* $\varphi \in \mathsf{SAT}$ *with length at least* N*, the output of* F *on* φ*, denoted by* $F(\varphi) = \psi 0 1^{|\psi|^{h(|\psi|)}}$*, satisfies the following: If* $|\psi| > N$*, then* $|\psi| < |\varphi|$*.*

Proof.(of claim) Since F runs in cn^k time, it follows that:

$$|\psi|^{h(|\psi|)} \quad < \quad |\psi| + 1 + |\psi|^{h(|\psi|)} \quad \leqslant \quad c|\varphi|^k$$

Thus,

$$|\psi| \quad < \quad c|\varphi|^{k/h(|\psi|)} \quad \leqslant \quad c|\varphi|^{1/2} \quad < \quad |\varphi|$$

The second and third inequalities come from the fact that $|\psi|, |\varphi| \geqslant N$. ∎

We now present a polynomial time algorithm for SAT, which contradicts the assumption that $\mathbf{NP} \neq \mathbf{P}$. On input φ, do the following.

- If $|\varphi| \leqslant N$, check by brute force if it is satisfiable. Otherwise, continue.
- Run F on φ, and let the output be $\psi 0 1^m$, for some m.
- Check if $m = |\psi|^{h(|\psi|)}$ by doing the following.
  1. Let $\ell = h(1^{|\psi|})$. (Recall that h is polynomial time computable.)
  2. Convert $|\psi|$ in its binary form and compute $|\psi|^\ell$ (in binary form).

3. Then, compare it with m.

- If $m \neq |\psi|^{h(|\psi|)}$, then REJECT immediately.
- Suppose $m = |\psi|^{h(|\psi|)}$.
  If $|\psi| \leqslant N$, check if ψ is satisfiable by brute force.
  If $|\psi| > N$, recursively call the algorithm on ψ. (Note that here $|\psi| < |\varphi|$.)

Each step in the algorithm takes polynomial time and the number of recursive call in this algorithm is at most $|\varphi|$. So, overall the algorithm runs in polynomial time. ∎

Next, consider the following lemma.

Lemma 3.3 *Suppose* $\mathbf{NP} \neq \mathbf{P}$. *If* $h : \mathbb{N} \to \mathbb{N}$ *is polynomial time computable (in unary representation) and bounded, i.e., there is a constant* c *such that* $h(n) \leqslant c$ *for every* n*, then* $\mathsf{SAT}_h \notin \mathbf{P}$.

Proof. Suppose $\mathsf{SAT}_h \in \mathbf{P}$. We will show that $\mathsf{SAT} \in \mathbf{P}$, which contradicts the assumption that $\mathbf{NP} \neq \mathbf{P}$. Consider the following algorithm. On input φ, do the following.

- Check if $\varphi 0 1^i \in \mathsf{SAT}_h$, for some $0 \leqslant i \leqslant |\varphi|^c$.
- ACCEPT iff there is i where $\varphi 0 1^i \in \mathsf{SAT}_h$.

∎

Combined with Lemmas 3.2 and 3.3, the following lemma implies Ladner's theorem, i.e., SAT_h is the desired intermediate **NP** language.

Lemma 3.4 *Suppose* $\mathbf{NP} \neq \mathbf{P}$. *There is a non-decreasing function* $h : \mathbb{N} \to \mathbb{N}$ *such that:*

- *h is polynomial time computable (in unary representation).*
- $\mathsf{SAT}_h \in \mathbf{NP}$.
- $\mathsf{SAT}_h \in \mathbf{P}$ *if and only if* h *is bounded.*

The function h for Lemma 3.4 is defined as follow. For every $n \geqslant 1$, the value $h(n)$ is determined by **Algorithm 1** below. Here $\mathcal{M}_i$ is the DTM whose encoding is the binary representation of i.

Algorithm 1

Input: 1^n, where $n \geqslant 1$.
Task: Compute $1^{h(n)}$.

1: **for** $i = 1, \dots, \log\log(n) - 1$ **do**
2: Let $\mathcal{M}_i$ be the i^{th} (1-tape) DTM.
3: **for all** $x \in \{0,1\}^*$ where $|x| \leqslant \log n$ **do**
4: Compute $\mathsf{SAT}_h(x)$ (i.e., recursively check if $x \in \mathsf{SAT}_h$).
5: Simulate $\mathcal{M}_i$ on x in $i|x|^i$ steps (using the UTM in Theorem 3.8).
6: **if** the results in lines 4 and 5 agree on all $x \in \{0,1\}^*$ where $|x| \leqslant \log n$ **then**
7: **return** i (in unary).
8: **return** $\log\log n$ (in unary).

2 Limit of diagonalization

A TM $\mathcal{M}$ with oracle access to a language K, denoted by $\mathcal{M}^K$, is a TM with a special tape called *oracle tape* and three special states $q_{\mathsf{query}}, q_{\mathsf{yes}}, q_{\mathsf{no}}$. Each time it is in q_{query}, it moves to q_{yes}, if $w \in K$ and to q_{no}, if $w \notin K$, where w is the string found in the oracle tape. In other words, when it is in q_{query}, the machine can "query" the membership of the language K. Regardless of the choice of K, such query counts only as one step. We denote by $L(\mathcal{M}^K)$ the language accepted by $\mathcal{M}^K$.

For a language K, we define the classes **P** and **NP** relativized to K as follows.

$$\mathbf{P}^K \stackrel{\mathsf{def}}{=} \{L : \text{there is a polynomial time DTM } \mathcal{M}^K \text{ such that } L(\mathcal{M}^K) = L\}$$
$$\mathbf{NP}^K \stackrel{\mathsf{def}}{=} \{L : \text{there is a polynomial time NTM } \mathcal{M}^K \text{ such that } L(\mathcal{M}^K) = L\}$$

Theorem 3.5 (Baker, Gill, Solovay 1975) *There is language A and B such that $\mathbf{P}^A = \mathbf{NP}^A$ and $\mathbf{P}^B \neq \mathbf{NP}^B$.*

Proof. For a **PSPACE**-complete language A, we can show that $\mathbf{P}^A = \mathbf{NP}^A$. (We will show in Lesson 2 that **PSPACE**-complete languages exist.)

To show the existence of B, we need the following notation. For a language $C \subseteq \{0,1\}^*$, define $\mathsf{unary}(C) \stackrel{\mathsf{def}}{=} \{1^n : \text{there is } w \in C \text{ with length } n\}$. Obviously, for every $C \subseteq \{0,1\}^*$, $\mathsf{unary}(C) \in \mathbf{NP}^C$.

The language B will be defined as $B \stackrel{\mathsf{def}}{=} \bigcup_{i\in\mathbb{N}} B_i$ where each B_i is a finite set defined inductively as follows. Each B_i is associated with an integer k_i such that $B_i = B \cap \{0,1\}^{\leqslant k_i}$. Here $\{0,1\}^{\leqslant k_i} \stackrel{\mathsf{def}}{=} \{w \in \{0,1\}^* : |w| \leqslant k_i\}$.

The base case is $B_0 = \emptyset$ and $k_0 = 0$. For the induction step, B_{i+1} is defined as follows, where we assume an enumeration of all oracle DTM $\mathcal{M}_0, \mathcal{M}_1, \ldots$.

- Let $n = k_i + 1$.
- Simulate oracle TM $\mathcal{M}_{i+1}$ on 1^n within $2^n/10$ steps.

  During the simulation $\mathcal{M}_{i+1}$ may query the oracle. For the query strings with length $\leqslant k_i$, the oracle answers are according to B_i. For the query strings with length $> k_i$, the oracle answers are "no."
- Let k_{i+1} be as follows.

$$k_{i+1} \stackrel{\mathsf{def}}{=} \begin{cases} n, & \text{if all the query strings has length } \leqslant k_i \\ m, & m \text{ is the maximal length of the query string with length } \geqslant n \end{cases}$$

- If $\mathcal{M}_{i+1}$ accepts 1^n within $2^n/10$ steps, we set $B_{i+1} \stackrel{\mathsf{def}}{=} B_i$.
- If $\mathcal{M}_{i+1}$ does not accept 1^n within $2^n/10$ steps, we set $B_{i+1} \stackrel{\mathsf{def}}{=} B_i \cup \{w\}$, where $w \in \{0,1\}^n$ and w is not one of the query strings.

From the definition of B, we can show that $\mathsf{unary}(B) \notin \mathbf{P}^B$. ∎

APPENDIX

A Universal Turing machines

Remark 3.6 For every k-tape TM $\mathcal{M}$ over input alphabet $\Sigma = \{0,1\}$, there is a k-tape TM $\mathcal{M}'$ over the same input alphabet $\Sigma = \{0,1\}$ and tape alphabet $\Gamma = \{0,1,\sqcup\}$ such that $L(\mathcal{M}) = L(\mathcal{M}')$. Moreover, if $\mathcal{M}$ runs in time/space $O(f(n))$, so does $\mathcal{M}'$.

Due to this, we always assume that the input and tape alphabet of Turing machines are $\Sigma = \{0,1\}$ and $\Gamma = \{0,1,\sqcup\}$, respectively.

Recall that $\lfloor \mathcal{M} \rfloor$ denotes the encoding of a TM $\mathcal{M}$.

Definition 3.7 A *Universal Turing machine* (UTM) is a k-tape DTM $\mathcal{U}$, for some $k \geqslant 1$, such that $L(\mathcal{U}) = \{\lfloor \mathcal{M} \rfloor \$ w \mid \mathcal{M} \text{ accepts } w \text{ and } w \in \{0,1\}^*\}$.

Theorem 3.8 *There is a UTM $\mathcal{U}$ such that for every DTM $\mathcal{M}$ and every word w, if $\mathcal{M}$ decides w in time t, then $\mathcal{U}$ decides $\lfloor \mathcal{M} \rfloor \$ w$ in time $(\alpha \cdot t \cdot \log t)$, where α does not depends $|w|$, but on size of the tape alphabet of $\mathcal{M}$ as well as the number of tapes and states of $\mathcal{M}$.*