CS 6810: Theory of Computing **Fall 2021**

Lecture 3: Sep 2, 2021

Lecturer: Eshan Chattopadhyay *Scribe: Dieqiao Feng*

1 NP-Completeness/Hardness

Recall: Assume $A, B \subseteq \{0,1\}^*$, $A \leq_p B$ if there exits a polynomial time computable function $f : \{0,1\}^* \to \{0,1\}^*$, such that $x \in A \Rightarrow f(x) \in B$ and $x \notin A \Rightarrow f(x) \notin B$.

Definition 1.1. *L is NP-hard if for any $L' \in NP$, $L' \leq_p L$. (Informally, L is as hard as any other problem in NP)*

Definition 1.2. *L is NP-complete if $L \in NP$ and $L \in$ NP-hard.*

Examples of NP-complete problems:

- **SAT**
  - Variables: $x_1, ..., x_n$
  - Literals: $x_1, \overline{x_1}, x_2, \overline{x_2}, ..., x_n, \overline{x_n}$
  - Clause: Disjunction (OR) of clauses
  - CNF: Conjunction (AND) of clauses, e.g.,

$$\phi(x_1, x_2, x_3, x_4) = (x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_2} \vee x_4)$$

  SAT $= \{\phi : \phi$ is a satisfiable CNF$\}$.

- **3-SAT**. 3-SAT is a SAT problem with the restriction that reach clause contains 3 literals.
- **Minimum Vertex Cover**. A vertex cover of a graph is a set of vertices that includes at least one endpoint of every edge of the graph. *Minimum Vertex Cover* is to find a minimum set of vertex cover.
- **Maximum Independent Set**. An *independent set* is a set of vertices in a graph, no two of which are adjacent. *Maximum Independent Set* is to find an independent set of largest possible size for a given graph $\mathcal{G}$.

Theorem 1.3. *Cook–Levin Theorem: 3-SAT is NP-complete.*

Sketch of the proof: Clearly SAT $\in$ NP since a proof that $\phi \in$ SAT is an assignment to the variables of ϕ making it true. To prove NP-hardness, for any problem L in NP we need to show $L \leq_p$ SAT. Now suppose that L can be solved by the non-deterministic Turning machine $M = (Q, \Sigma, s, F, \delta)$, where Q is the set of states, Σ is the alphabet of tape symbols, $s \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states, and $\delta \subseteq ((Q \setminus F) \times \Sigma) \times (Q \times \Sigma \times \{-1, +1\})$ is the transition relation. Suppose further that M accepts or rejects an instance of L in time at most $p(n)$ where n is the size of the instance and p is a polynomial.

Now the main part of the proof is to show that for each input instance x, we construct a Boolean expression which is satisfiable if and only if the verifier Turning machine M_x that accepts

Algorithm 1: Algorithm for P_1

```
Input: x
if A(x, ε) → NO then
    return No;
z ← ε;
while (x, z) ∉ R do
    if A(x, z ∘ 0) → YES then
        z ← z ∘ 0;
    else
        z ← z ∘ 1;
return z;
```

a polysized witness y (if $x \in L$). We try to give a very high level overview here: since M runs in polynomial time, so at most polynomially tape cells are ever accessed. So we can define Boolean variables capturing the contents of the relevant tape cells at every step of computation. Moreover, we can also define the state s and head location of M using Boolean variables. The fact that every step of computation is local allows us to encode TAPE/HEAD/STATE transition between steps using Boolean formulas. For more details about how computations are encoded, please refer to the Arora-Barak book.

Two steps of proving NP-completeness of L:

1. $L \in$ NP
2. $H \in$ NP-hard and $H \leq_p L$

2 Search vs Decision Problems

We first define an NP relation. A binary relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ is in NP if

1. $(x, y) \in R \Rightarrow |y| \leq C \cdot |x|^C$, for some constant C
2. A polynomial time verifier V exists, such that $V(x, y) = 1$ if and only if $(x, y) \in R$

Corresponding to R, we have:

- Decision problem: Input x. Output yes if and only if $\exists y$ s.t. $(x, y) \in R$.
- Search problem: Input x. Output y (if any) s.t. $(x, y) \in R$.

Claim 2.1. *For any NP search problem P_1, there is a corresponding NP-decision problem P_2, such that if P_2 can be computed in time $T(n)$, then P_1 can be computed in time poly$(n) \cdot T($poly$(n))$.*

Corollary 2.2. *If $P = NP$, then search NP has polynomial time algorithm.*

Proof of Claim 2.1: Let R be the NP relation corresponding to P_1, and P_2 accepts (x, y) if there is $z \in \{0,1\}^*$, such that $(x, y \circ z) \in R$. Let $\mathcal{A}(\cdot, \cdot)$ be the algorithm for P_2. See Algorithm 1 for the description for P_1.

3 Diagonalization and Applications

Diagonalization is a useful and powerful argument that has found many applications, ranging from the proof that infinity of real numbers is larger than natural numbers to proving that the Halting problem is undecidable. We will see here a couple of more interesting applications of this argument.

3.1 Time hierarchy theorem

A natural question to consider is if strictly richer classes of languages can be computed by Turing machines given more compute time. This indeed turns out to be the case, and is called as the Time Hierarchy theorem, which we state and prove below. The proof will be based on a simple diagonalization argument.

Theorem 3.1. *For any "time-constructible" $t(n)$ there is a language $L \in DTIME(t(n)^3)$ but any turning machine takes at least $t(n)$ time to compute L.*

This theorem informally says that given more time, a Turing machine can solve more problems, and so the time-bounded hierarchy of complexity classes does not completely collapse.

Proof: Define the language of the encodings of machines and their inputs which cause them to half within $t(n)$

$$L = \{(\llcorner M \lrcorner, x) : M \text{ accepts } x \text{ in } t(|x|) \text{ steps}\}.$$

Now we can decide L by way of a deterministic Turning machine R, that simulates M for $t(|x|)$ steps. It is safe to say that this takes at most $f(m)^3$ operations, where m is the size of $\llcorner M \lrcorner$ and x. So

$$L \in \text{TIME}(t(m)^3).$$

The rest of the proof will show that

$$L \notin \text{TIME}(t(m)).$$

Assume $L \in \text{TIME}(t(m))$ and let K be the decision machine for L. We construct another machine N, which takes a machine description $\llcorner M \lrcorner$ and runs K on the tuple $(\llcorner M \lrcorner, \llcorner M \lrcorner)$, and then accepts if K rejects, and rejects if K accepts. Now feed $\llcorner N \lrcorner$ into N itself and ask the question whether N accepts its own description as input, we get:

- If N accepts $\llcorner N \lrcorner$, this means K rejects $(\llcorner N \lrcorner, \llcorner N \lrcorner)$, so $(\llcorner N \lrcorner, \llcorner N \lrcorner)$ is not in L, this implies N does not accept $\llcorner N \lrcorner$ in $t(n)$ steps. Contradiction.
- If N rejects $\llcorner N \lrcorner$, this means K accepts $(\llcorner N \lrcorner, \llcorner N \lrcorner)$, so $(\llcorner N \lrcorner, \llcorner N \lrcorner)$ is in L, this implies N does accept $\llcorner N \lrcorner$ in $t(n)$ steps. Contradiction.

3.2 Ladner's theorem

Here is another interesting application of diagonalization.

Theorem 3.2. *Suppose $P \neq NP$, $\exists L \in NP \setminus P$ which is not NP-complete.*

We will prove this in the next class.