**CS 39: Theory of Computation** **Winter '23**

# PSET 7 — 02/06/2023

***Prof. Chakrabarti*** ***Student: Amittai Siavava***

## Credit Statement

I discussed ideas for this homework assignment with Paul Shin. I also Boxian's office hours wherein we discussed some general approaches to some of the problems. I referred to the following books:

(a) **Introduction to the Theory of Computation** by **Michael Sipser**.

## Problem 1.

Show that one of the languages $\mathsf{EQ}_{\mathsf{CFG}}$ and $\overline{\mathsf{EQ}_{\mathsf{CFG}}}$ is recognizable, whereas the other is unrecognizable. You may assume that $\mathsf{ALL}_{\mathsf{CFG}}$ is unrecognizable.

Let's fix our alphabet to $\Sigma = \{0, 1\}$.

(a) $\mathsf{EQ}_{\mathsf{CFG}}$ is unrecognizable. We know that $\mathsf{ALL}_{\mathsf{CFG}}$ is unrecognizable. Consider the following reduction of $\mathsf{ALL}_{\mathsf{CFG}}$ to $\mathsf{EQ}_{\mathsf{CFG}}$:

$T$ = "On input $\langle M \rangle$, where $M$ is a CFG:

1. Construct PDA $P$ as follows:

   $P$ = "On input $\langle w \rangle$:

   1. If $w \in \Sigma^*$, Accept .
   2. Otherwise, Reject ."

2. Construct CFG $G$ equivalent to $P$, such that $\mathcal{L}(G) = \mathcal{L}(P) = \Sigma^*$.
3. Return $\langle M, G \rangle$. "

$$M \in \mathsf{ALL}_{\mathsf{CFG}} \iff M \text{ accepts all strings in the input alphabet.}$$
$$\iff \mathcal{L}(M) = \Sigma^*$$
$$\iff \langle M, G \rangle \in \mathsf{EQ}_{\mathsf{CFG}}$$

Since $\mathsf{ALL}_{\mathsf{CFG}}$ is unrecognizable, $\mathsf{EQ}_{\mathsf{CFG}}$ must also be unrecognizable.

(b) $\overline{\mathsf{EQ}_{\mathsf{CFG}}}$ is recognizable.

Let $G_1$ and $G_2$ be two CFGs defined over the same alphabet $\Sigma$ and $M_1$ and $M_2$ be the equivalent PDAs such that $\langle G_1, G_2 \rangle \in \overline{\mathsf{EQ}_{\mathsf{CFG}}}$. This means $\mathcal{L}(M_1) \neq \mathcal{L}(M_2)$, so there exists *at least* one string in $x \in \Sigma^*$ such that $x \in \mathcal{L}(M_1)$ or $x \in \mathcal{L}(M_2)$, but not both (equivalently, $x \in \mathcal{L}(M_1) \cup \mathcal{L}(M_2)$ and $x \notin \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$).

Since $\Sigma^*$ is decidable, we can construct a TM $E$ to list members of $\Sigma^*$. in lexicographic order.

We can then recognize $\overline{\mathsf{EQ}_{\mathsf{CFG}}}$ by the following TM $T$:

$T$ = "On input $\langle M_1, M_2 \rangle$:

1. Run $E$ to enumerate members of $\Sigma^*$.
2. As each string is listed:
   - **2.1.** Simulate $M_1$ on $x$ and $M_2$ on $x$.
   - **2.2.** If $M_1$ accepts $x$ and $M_2$ rejects $x$ or $M_1$ rejects $x$ and $M_2$ accepts $x$, Accept .
   - **2.3.** Otherwise, continue enumerating strings and testing them."

If an input $\langle M_1, M_2 \rangle$ is in $\overline{\mathsf{EQ}_{\mathsf{CFG}}}$, then $T$ will eventually halt and accept the input when it encounters a string not accepted by both $M_1$ and $M_2$ r rejected by both $M_1$ and $M_2$.

However, if an input $\langle M_1, M_2 \rangle$ is not in $\overline{\mathsf{EQ}_{\mathsf{CFG}}}$, then $T$ will never halt since $\Sigma^*$ is infinite.

**Problem 2.**

Thanks to the Cooke-Levin theorem, we know that 3Color can be polynomial-time reduced to 3SAT. However, for this problem, pretend that you did not know that theorem. Give a direct polynomial-time reduction from 3Color to 3SAT.

Let $G = (V, E)$ be the graph in question. Let $R, G, B$ be predicates saying a given vertex is colored red, green, or blue, respectively, e.g. $Rx$ means "vertex $x$ is colored red". To model 3Color as a 3SAT instance, we first need to define axioms for the conditions that need to be satisfied in 3Color, ensuring that each clause has at most 3 literals.

(i) Each vertex $v \in V$ must be assigned a color:

$$\Sigma_1 = \{(Rv \vee Gv \vee Bv) : v \in V\}$$

(ii) Each vertex may not be assigned more than one color (equivalently, each vertex *must* not be assigned 2 of the 3 possible colors):

$$\Sigma_{2.1} = \{(\neg Rv \vee \neg Gv \vee \neg Gv) : v \in V\}$$

$$\Sigma_{2.2} = \{(\neg Rv \vee \neg Bv \vee \neg Bv) : v \in V\}$$

$$\Sigma_{2.3} = \{(\neg Gv \vee \neg Bv \vee \neg Bv) : v \in V\}$$

$$\Sigma_2 = \Sigma_{2.1} \cup \Sigma_{2.2} \cup \Sigma_{2.3}$$

(iii) Adjacent vertices are not colored the same color:

$$\Sigma_{3.1} = \{(\neg Rv \vee \neg Ru \vee \neg Ru) : (u, v) \in E\}$$

$$\Sigma_{3.2} = \{(\neg Gv \vee \neg Gu \vee \neg Gu) : (u, v) \in E\}$$

$$\Sigma_{3.3} = \{(\neg Bv \vee \neg Bu \vee \neg Bu) : (u, v) \in E\}$$

$$\Sigma_3 = \Sigma_{3.1} \cup \Sigma_{3.2} \cup \Sigma_{3.3}$$

Finally, let $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$, and define the formula

$$\psi = \bigwedge_{\sigma \in \Sigma} \sigma$$

We can thus define a reduction from 3Color to 3SAT as follows:

$T$ = "On input $G = (V, E)$, where $G$ is a Graph:

**1.** Construct the formula $\psi$ as described above.

**2.** Output $\psi$."

$$
\begin{aligned}
G \in \text{3Color} &\iff \exists \text{ a valid 3-coloring of } G \\
&\iff \exists \text{ a coloring of } G \text{ satisfying } \psi \\
&\iff \psi \text{ is satisfiable} \\
&\iff \psi \in \text{3SAT}
\end{aligned}
$$

**Problem 3.**

Sloppy Q. Thinker and Fawlty J. Logician are two computer science amateurs who have not taken a rigorous course in the Theory of Computation, such as Dartmouth's CS 39. They have come up with the following fallacious "proofs."

Criticize each of these fallacious "proofs" thoroughly. How would you convince Sloppy and Fawlty that not only are they wrong, but ideas along these lines simply cannot work for proving what they are claiming?

(a) Sloppy has this "proof" that $P \neq NP$. Consider an algorithm for SAT : "On input $\varphi$, try all possible assignments to the variables. Accept if any satisfy $\varphi$." This algorithm clearly requires exponential time. Thus, SAT has exponential time complexity. Therefore SAT is not in $P$. Because SAT is in $NP$, it must be the case that $P \neq NP$.

(b) Fawlty instead has a "proof" that $P = NP$. Take an arbitrary language $l \in NP$. Let $N$ be a polynomial-time NDTM decider for $L$. Since NDTMs are equivalent to TMs, there is a deterministic TM, $M$, that also decides $L$. This shows that $L \in P$. Therefore $NP \subseteq P$. Since we clearly have $P \subseteq NP$, it follows that $P = NP$.

**Problem 4.**

Let DoubleSAT = $\{\langle\varphi\rangle : \varphi$ is a cnf with at least 2 satisfying assignments$\}$. Prove that DoubleSAT is NP-complete.

**Problem 5.**

Prove that if $P$ were equal to $NP$, then there would exist an efficient (i.e. deterministic polynomial-time) algorithm to break the RSA cryptosystem.

Hint: You can solve this problem even if you haven't studied the RSA cryptosystem before. Here is all you need to know about RSA. The RSA scheme involves a modulus of the form $N = pq$ where $p$ and $q$ are $n$-bit prime numbers, for some large $n$. The value of $N$ is known to the world, but the factorization into $p$ and $q$ is secret. If an adversary figures out the factorization, they can decrypt any messages that were encrypted with the modulus $N$.

Given an integer $N$ that is guaranteed to be the product of two primes $p$ and $q$, the problem of finding $p$ and $q$ is in the class NP as it can be solved non-deterministically with the following NDTM:

$T$ = "On input $N$, where $N = pq$ for some (unknown) primes $p$ and $q$:

1. Non-deterministically guess two integers $p \in (1, N)$ and $q \in (1, N)$.
2. If $p \cdot q = N$, Return $\langle p, q \rangle$."

Thus, $T$ *non-deterministically* finds the factorization of $N$ in polynomial time. If $P = NP$, then there exists some *deterministic* turing machine that can also find $p$ and $q$ in polynomial time, hence determine the factorization for any RSA number $N$ and, using the factorization, decrypt any RSA-encrypted message encrypted with the modulus $N$.

**Problem 6.**

Recall that the theory of time complexity was developed using worst case cost: for a particular length $n$, we used the maximum of all time costs for inputs of length up to $n$. Why not the average? This exercise will show you one good reason why not: the natural notion of "average" fails to properly capture the hardness of a computational problem.

Let $M$ be a decider TM with input alphabet $\Sigma$. Recall that we defined $\textsc{TimeCost}_M(x)$, for $x \in \Sigma^*$, to be the number of steps $M$ takes on input $x$ until it halts. Let us now define the function $\textsc{AvgTimeCost}_M : \mathbb{N} \to \mathbb{N}$ as follows:

$$\textsc{AvgTimeCost}_M(n) = \frac{1}{|\Sigma|^n} \sum_{x \in \Sigma : |x|=n} \textsc{TimeCost}_M(x).$$

The corresponding class of algorithms that run in "polynomial time on average" would be

$$\mathsf{AvgP} = \left\{ L \subseteq \Sigma^* : \exists \text{ decider } M \text{ and integer } k \text{ such that } \mathcal{L}(M) = L \text{ and } \textsc{AvgTimeCost}_M(n) = O(n^k) \right\}.$$

Construct a language $L$ such that on the one hand $L$ is $NP$-complete, (strongly suggesting that it is hard) and, on the other hand, $L \in \mathsf{AvgP}$ (misleadingly suggesting that it is easy).

Hint: Suitably modify the langauge SAT .

Consider the language

$$\textsc{WeirdSAT} = \left\{ x0^{|x|} : x \in \text{SAT} \right\}.$$

To see that WeirdSAT is NP-complete, consider the following reduction of SAT to WeirdSAT , that takes well-formed strings in WeirdSAT and returns the corresponding prefix (that is required to be in SAT ):

$T_0$ = "On input $\langle w0^{|w|} \rangle$:

**1.** Return $\langle w \rangle$."

Clearly, $w \in \text{SAT} \iff w0^{|w|} \in \textsc{WeirdSAT}$. Since SAT is NP-complete, WeirdSAT must also be NP-complete since if we could efficiently solve WeirdSAT we could use the algorithm to derive efficient solutions to SAT .

Now, consider the average-case runtime costs of any recognizer $M$ for WeirdSAT on inputs $x$ in WeirdSAT . We know SAT is NP-complete, let $T_{\text{SAT}}$ be an NDTM that solves SAT .

$T_1$ = "on input $\langle w \rangle$:

1. Determine if $w$ is of the form $x0^{|x|}$ for some $x \in \Sigma^*$. If not, Reject .
2. If yes, write $w$ as$x0^{|x|}$, and run $T_{\text{SAT}}$ on input $\langle x \rangle$.
3. If $T_{\text{SAT}}$ accepts, Accept , otherwise Reject ."

Now let's consider the running time of $T_1$. Let $n$ be the size of input strings. Step one determines if the string can be broken down into two equal parts, with the second part being all zeros. This can be done in $O(n)$ time as it only requires scanning the string a constant number of times. Step 2 then tests if the prefix $x$ is in SAT . This requires testing all possible assignments to the variables in $x$, since SAT is NP-complete. Thus, this takes $2^{|x|} = 2^{n/2} = O(2^n)$ time.

Now, let's consider the average-case running time of $T_1$ on strings of fixed lengths $n$.

(i) $n$ is odd: Clearly, a string with odd length cannot be written as two strings of equal length. Therefore, $T_1$ only needs to run the checker for step 1, which takes $O(n)$ time.

(ii) $n$ is even: If a string of length $n$ is in WeirdSAT then its second half must be of the form $0^{n/2}$. Given an alphabet of size $m$, there are a total of $m^n$ strings of length $n$, and only $m^{n/2}$ strings are plausible members of WeirdSAT (contingent on checking that the prefix is in SAT ). Thus, the running time will be $O(2^{n/2}) = O(2^n)$ for *each* of the $m^{n/2}$ strings, and $O(n)$ for the remaining $m^n - m^{n/2}$ strings. Thus;

$$
\begin{aligned}
\textsc{AvgTimeCost}_M(n) &= \frac{1}{|\Sigma|^n} \sum_{x \in \Sigma : |x|=n} \textsc{TimeCost}_M(x) \\
&= \frac{(m^n - m^{n/2})n + m^{n/2}2^{n/2}}{m^n} \\
&= \frac{(m^n - m^{n/2})}{m^n} n + \frac{m^{n/2}}{m^n} 2^{n/2} \\
&= n - \frac{m^{n/2}}{m^n} n + \frac{m^{n/2}}{m^n} 2^{n/2} \\
&= O(n) + \left(\frac{2}{m}\right)^n \\
m \geq 2 &\Rightarrow \textsc{AvgTimeCost}_M(n) = O(n)
\end{aligned}
$$