

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 EQ_{CFG} and $\overline{\text{EQ}_{\text{CFG}}}$ is recognizable, whereas the other is unrecognizable. You may assume that ALL_{CFG} is unrecognizable.

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

- (a) EQ_{CFG} is unrecognizable. We know that ALL_{CFG} is unrecognizable. Consider the following reduction of ALL_{CFG} to EQ_{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 $\langle M, G \rangle$."

$$M \in \text{ALL}_{\text{CFG}} \iff M \text{ accepts all strings in the input alphabet.}$$

$$\iff \mathcal{L}(M) = \Sigma^*$$

$$\iff \langle M, G \rangle \in \text{EQ}_{\text{CFG}}$$

Since ALL_{CFG} is unrecognizable, EQ_{CFG} must also be unrecognizable.

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

Let G_1 and G_2 be two CFGs defined over the same alphabet Σ and M_1 and M_2 be the equivalent PDAs such that $\langle G_1, G_2 \rangle \in \overline{\text{EQ}_{\text{CFG}}}$. 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 Σ^* is decidable, we can construct a TM E to list members of Σ^* in lexicographic order.

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

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

1. Run E to enumerate members of Σ^* .
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 $\overline{\text{EQ}_{\text{CFG}}}$, then T will eventually halt and accept the input when it encounters a string not accepted by both M_1 and M_2 or rejected by both M_1 and M_2 .

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

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

This construction of Σ_1 can be done in $O(|V|)$ time.

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

$$\Sigma_{2,1} = \{(\neg Rv \vee \neg Gv \vee \neg Gv) : v \in V\}$$

$$\Sigma_{2,2} = \{(\neg Rv \vee \neg Bv \vee \neg Bv) : v \in V\}$$

$$\Sigma_{2,3} = \{(\neg Gv \vee \neg Bv \vee \neg Bv) : v \in V\}$$

$$\Sigma_2 = \Sigma_{2,1} \cup \Sigma_{2,2} \cup \Sigma_{2,3}$$

This construction of Σ_2 can be done in $O(|V|)$ time.

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

$$\Sigma_{3,1} = \{(\neg Rv \vee \neg Ru \vee \neg Ru) : (u, v) \in E\}$$

$$\Sigma_{3,2} = \{(\neg Gv \vee \neg Gu \vee \neg Gu) : (u, v) \in E\}$$

$$\Sigma_{3,3} = \{(\neg Bv \vee \neg Bu \vee \neg Bu) : (u, v) \in E\}$$

$$\Sigma_3 = \Sigma_{3,1} \cup \Sigma_{3,2} \cup \Sigma_{3,3}$$

This construction of Σ_3 can be done in $O(|E|)$ time.

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

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

This construction of ψ can be done in $O(|V| + |E|)$ time.

We can thus define a reduction from 3COLOR to 3SAT in $O(|V| + |E|)$ time as follows:

“On input $G = (V, E)$, where G is a graph:

1. Construct the formula ψ from G as described above.
2. Output ψ .”

$$G \in \text{3COLOR} \iff \exists \text{ a valid 3-coloring of } G$$

$$\iff \exists \text{ a coloring of } G \text{ satisfying } \psi$$

$$\iff \psi \text{ is satisfiable}$$

$$\iff \psi \in \text{3SAT}$$

Problem 3.

Sloppy Q. Thinker and Fawltly 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 Fawltly 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 φ , try all possible assignments to the variables. Accept if any satisfy φ ." 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 $P \neq NP$.

Sloppy's solution overlooks the fact that $P \subseteq NP$. Many (perhaps all?) problems that can be solved in polynomial time by some clever algorithm can also be solved in exponential time by a brute-force algorithm. For instance, finding the minimum value in a *sorted* array can be solved efficiently in $O(1)$ time by using the fact that the minimum value is the first element, yet the same problem can also be solved very inefficiently in $O(n^n)$ time by non-deterministically guessing a possible minimum value's index and comparing the value at that index to every other value in the array. Sloppy needs to take the necessary step of showing that SAT cannot, in fact, be solved in polynomial time, such as by reducing a problem that has been proven to be NP-complete to SAT hence showing that SAT must not be in P (hence, $P \neq NP$).

- (b) Fawltly 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 $P = NP$.

Fawltly assumes that the equivalent deterministic TM M also runs in polynomial-time. This is not necessarily the case — in fact, the direct equivalent would run in exponential time. If indeed $L \in P$, Fawltly needs to directly prove the existence of a polynomial-time *deterministic* TM that decides L .

Problem 4.

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

DOUBLESAT is in NP: Given a cnf φ , we can check if φ is in DOUBLESAT using a non-deterministic TM T as follows:

T_1 = “On input $\langle \varphi \rangle$, where φ is a cnf:

1. Guess two assignments A_1 and A_2 to the variables in φ .
2. If $A_1 \neq A_2$ and both A_1 and A_2 satisfy φ , ACCEPT.”

DOUBLESAT is NP-complete. Consider the following reduction of SAT to DOUBLESAT :

T_2 = “On input $\langle \varphi \rangle$, where φ is a cnf:

1. Define a new variable γ that does not occur in φ .
2. Construct a new cnf $\varphi' = \varphi \wedge (\gamma \vee \neg\gamma)$.
3. RETURN $\langle \varphi' \rangle$.”

$\varphi \in \text{SAT} \iff \exists$ at least one assignment that satisfies φ (let A be one such assignment).

$\iff \exists$ at least two assignments that satisfy φ' , $B = A \cup \{\gamma = T\}$, and $C = A \cup \{\gamma = F\}$.

$\iff \varphi' \in \text{DOUBLESAT}$

SAT is NP-complete, and so must DOUBLESAT be. If it were not, then clearly having an efficient algorithm for DOUBLESAT would give a plausible and efficient way to derive solutions to SAT, contradicting the fact that SAT is proven to be 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. but 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 Σ . Recall that we defined $\text{TIMECOST}_M(x)$, for $x \in \Sigma^*$, to be the number of steps M takes on input x until it halts. Let us now define the function $\text{AVGTIMECOST}_M : \mathbb{N} \rightarrow \mathbb{N}$ as follows:

$$\text{AVGTIMECOST}_M(n) = \frac{1}{|\Sigma|^n} \sum_{x \in \Sigma^n} \text{TIMECOST}_M(x).$$

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

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

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, $L \in \text{AvgP}$ (misleadingly suggesting that it is easy).

Hint: Suitably modify the language SAT .

Consider the language

$$\text{WEIRDSAT} = \{x0^{|x|} : x \in \text{SAT}\}.$$

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, $w \in \text{SAT} \iff w0^{|w|} \in \text{WEIRDSAT}$. 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_{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_{SAT} on input $\langle x \rangle$.
3. If T_{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 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;

$$\begin{aligned}
 \text{AVGTIMECOST}_M(n) &= \frac{1}{|\Sigma|^n} \sum_{x \in \Sigma : |x|=n} \text{TIMECOST}_M(x) \\
 &= \frac{(m^n - m^{n/2})n + m^{n/2}2^{n/2}}{m^n} \\
 &= \frac{(m^n - m^{n/2})}{m^n}n + \frac{m^{n/2}}{m^n}2^{n/2} \\
 &= O(n) + \frac{(2m)^{n/2}}{m^n} \\
 &= O(n) + \left(\frac{2}{m}\right)^n
 \end{aligned}$$

$$m \geq 2 \Rightarrow \text{AVGTIMECOST}_M(n) = O(n)$$