CS 4510 Automata and Complexity 4/15/2024

Homework 7: Complexity (medium mode)

Vidit D. Pokharna *Due: 4/29/2024*

You should submit a typeset or *neatly* written pdf on Gradescope. The grading TA should not have to struggle to read what you've written; if your handwriting is hard to decipher, you will be asked to typeset your future assignments. Four bonus points if you use LaTeX, and our template. You may collaborate with other students, but any written work should be your own. **Many of these problems have two sentence answers if you can apply the theorems proved in class**.

1. (5 points) Prove that every PSPACE-hard problem is also NP-hard
   We know that NP $\subseteq$ PSPACE. Given that a PSPACE-hard problem is challenging at least to the same extent as the most challenging problems within NP, we deduce that if all problems in NP can be transformed into a PSPACE-hard problem with polynomial-time computations, then every problem in NP can also be transformed in polynomial time to any given PSPACE-hard problem.

   Consider an arbitrary problem that is hard for PSPACE, denoted as Q. Since every problem in NP, which is a more restrictive class than PSPACE, can be converted to Q in polynomial time (by the definition of Q being PSPACE-hard), it follows that Q is at least as difficult to solve as the hardest problems in NP. This is essentially what being NP-hard means – being at least as tough as any problem in NP. Thus, through this logical chain, we conclude that if Q is hard for PSPACE, it must be hard for NP as well. Hence, all PSPACE-hard problems must indeed be NP-hard.

2. We proved that $TQBF$ is PSPACE-complete. We can use this to give a second proof that PSPACE = NPSPACE, by proving that $TQBF$ is also a NPSPACE complete problem. For this problem, **do not assume** PSPACE = NPSPACE by Savitch's theorem, as that is what we are trying to prove, you may only assume PSPACE $\subseteq$ NPSPACE.

   (a) (2 points) Prove that $TQBF \in$ NPSPACE (Hint, you should not be able to show that $TQBF \in$ NP).

   NPSPACE is the class of problems solvable by a non-deterministic Turing machine using polynomial space. The complexity class NP, by contrast, is defined by problems solvable by non-deterministic Turing machines in polynomial time, not space, which is a distinct concept.

   $TQBF$ involves determining the truth of a fully quantified Boolean formula. To solve $TQBF$ within NPSPACE, a non-deterministic Turing machine would proceed by choosing values for the existential quantifiers non-deterministically. It's important to note that while the machine can guess values for existential quantifiers, it cannot do the same for universal quantifiers by definition. However, for each universally quantified variable, the machine can check all possible values within polynomial space since it can reuse the space after each check. The machine does not require more space than the length of the formula plus some polynomial amount for the computation's variables and state.

   Given that NPSPACE contains all decision problems that a non-deterministic Turing machine can solve using polynomial space, and since we can design such a non-deterministic algorithm for $TQBF$, we can conclude that $TQBF \in$ NPSPACE. This argument is based on the space-bounded computation rather than time-bounded, avoiding the incorrect assertion that a non-deterministic Turing machine can guess universally quantified variables.

(b) (3 points) We proved that $\forall L \in \mathsf{PSPACE}$ that $L \leq_p TQBF$. Explain how you could modify the proof we did to show that $\forall L \in \mathsf{NPSPACE}$ that $L \leq_p TQBF$ (Hint, why could we show SAT complete for a nondeterministic class, and $TQBF$ complete for a deterministic one?)

Given that $TQBF$ is PSPACE-complete, we know that for any language L in PSPACE, $L \leq_p TQBF$. Since we assume that PSPACE $\subseteq$ NPSPACE, this implies that any language L in NPSPACE can also be reduced to $TQBF$ within polynomial time.

To modify our proof to show that $L \leq_p TQBF$ for every language L in NPSPACE, we can consider the non-deterministic Turing machine that decides L using polynomial space. A polynomial-time reduction can be constructed from L to an instance of $TQBF$, leveraging the fact that a non-deterministic computation within polynomial space can be simulated by a deterministic machine within the same space bounds.

The completeness of SAT for the non-deterministic class (NP) and the completeness of $TQBF$ for the deterministic space-bounded class (PSPACE) provide the framework for this argument. Because non-deterministic space-bounded computations can be simulated deterministically with the same space complexity, the reduction to $TQBF$ is valid for languages in NPSPACE, thus showing that $TQBF$ is NPSPACE-hard.

Therefore, we can conclude without invoking Savitch's theorem that if a language is in NPSPACE, it can be reduced in polynomial time to $TQBF$, which supports the proof that PSPACE = NPSPACE under our initial assumption.

3. (5 points) Prove that $\mathsf{P} = \mathsf{PSPACE}$ has no relativizing proof.

Relativizing proofs are those that remain valid when both sides of the equation $\mathsf{P} = \mathsf{PSPACE}$ are augmented with the same oracle A, resulting in $\mathsf{P}^A = \mathsf{PSPACE}^A$. The existence of an oracle that preserves an equality would indicate that the proof method is relativizing since it hinges on properties that are not affected by the oracle's addition.

However, we know that there exist languages for which $\mathsf{P}^A = \mathsf{NP}^A$ and other languages for which $\mathsf{P}^A \neq \mathsf{NP}^A$. This indicates that relativizing methods cannot resolve the question of whether P equals NP because the existence of an oracle can change the answer.

If we were to assume that there is a relativizing proof that $\mathsf{P} = \mathsf{PSPACE}$, we would expect this proof to work even when we add the same oracle A to both sides, implying $\mathsf{P}^A = \mathsf{PSPACE}^A$. If the classes were equal under relativization, then adding the oracle A would also imply $\mathsf{NP}^A = \mathsf{NPSPACE}^A$, which would in turn imply $\mathsf{P}^A = \mathsf{NP}^A$ since $\mathsf{PSPACE} = \mathsf{NPSPACE}$ and $\mathsf{P} = \mathsf{PSPACE}$.

But we already know from the relativization barrier that for some oracles, P^A does not equal NP^A, and for others, it does. This contradiction shows that the assumption that $\mathsf{P} = \mathsf{PSPACE}$ has a relativizing proof cannot hold. Therefore, we conclude that any proof of $\mathsf{P} = \mathsf{PSPACE}$ cannot be a proof that relativizes.

Thus, without assuming the existence of any particular oracle or making similar assumptions as done in the paper, we've deduced that any relativizing proof of $\mathsf{P} = \mathsf{PSPACE}$ would lead to a contradiction with known results about relativization and oracle separations. This indicates that there cannot be a relativizing proof of $\mathsf{P} = \mathsf{PSPACE}$.

4. (5 points) Prove by diagonalization that $\mathsf{P} \neq \mathsf{EXP}$.

The Time Hierarchy Theorem can be utilized to construct a new problem that is outside a given complexity class by ensuring that it differs from the computation of each machine in the class on at least one input.

Begin by enumerating all possible Turing machines that run in polynomial time $O(n^k)$ for some constant k. Since the set of all Turing machines is countable, this enumeration is possible.

Construct a language L such that for every Turing machine M_i in the enumeration, there exists an input x_i on which M_i fails to decide whether x_i is in L. This can be done by designing L so that for each M_i and input x_i of length n, if M_i accepts x_i in polynomial time, then x_i is not in L, and if M_i does not accept x_i, then x_i is in L. This ensures that L is different from the language decided by M_i for at least one input, specifically x_i.

According to the Time Hierarchy Theorem, there is a Turing machine M that decides L in time $O(2^n)$ but not in $O(\frac{2^n}{n})$. Since every polynomial is $O(\frac{2^n}{n})$, this M cannot be one of the polynomial-time machines that was enumerated. Hence, L is a language in EXP (because it is decided by M in exponential time) but not in P.

Since L can be decided in exponential time but not in polynomial time, it is shown that EXP contains languages that are not in P. This establishes that $\mathsf{P} \neq \mathsf{EXP}$.

This proof by diagonalization demonstrates that there will always be problems that can be solved in exponential time that are not solvable by any Turing machine in polynomial time, reflecting the non-equivalence of the complexity classes P and EXP.

5. (5 points) Prove for all oracles A that $\mathsf{P}^A \neq \mathsf{EXP}^A$. Proof your proof of the previous relativizes.

Instead of considering standard Turing machines, consider Turing machines with access to an oracle A. These machines can make queries to A during their computation at no additional time cost.

Enumerate all oracle Turing machines that run in polynomial time with access to oracle A. This enumeration is still countable, even with the oracle access.

Construct a language L^A such that for every oracle Turing machine M_i^A in our enumeration, there exists an input x_i on which M_i^A fails to decide whether x_i is in L^A. Ensure that L^A is designed so that it differs from the language decided by M_i^A on input x_i.

By the Time Hierarchy Theorem, which now considers oracle machines, there exists an oracle Turing machine M^A that decides L^A in time $O(2^n)$ but not in $O(\frac{2^n}{n})$. The Turing machine M^A does not belong to the enumeration of oracle Turing machines that run in polynomial time since the theorem guarantees the existence of problems that require more than polynomial time (with the same oracle A).

The language L^A can be decided in exponential time but not in polynomial time with respect to the oracle A. Hence, it is shown that EXP^A contains languages that are not in P^A, for any oracle A.

The reason this proof relativizes is that the diagonalization argument used to prove the Time Hierarchy Theorem doesn't depend on the specifics of the oracle A; the construction of the diagonal language L^A and the fact that there are problems outside of P^A that can be decided in EXP^A still hold when the Turing machines are given access to A. The core argument of the Time Hierarchy Theorem remains valid when an oracle is introduced because the separation between complexity classes is a function of the growth rates of their respective time bounds, which is preserved even when an oracle is added.

Therefore, the theorem and its proof method hold true generally and relativize effectively, as the presence of an oracle doesn't alter the fundamental time complexity relationships exploited by the theorem. This is why for any oracle A, P^A is strictly smaller than EXP^A, and therefore $\forall$ oracles $A, \mathsf{P}^A \neq \mathsf{EXP}^A$.

6. (5 points) Prove that if then there is an oracle A such that $\mathsf{TIME}^A(2^n/\log n) \neq \mathsf{NP}^A$.

Enumerate all polynomial-time oracle Turing machines $M_1^A, M_2^A, \ldots$ that decide languages in $\mathsf{TIME}^A(2^n/\log n)$.

Construct a language L^A to diagonalize against all machines in $\mathsf{TIME}^A(2^n/\log n)$. For each string 1^n, decide whether it should be in L^A based on the behavior of the oracle machines. This language will be defined such that for each Turing machine M_i^A, there is a string 1^n for which M_i^A cannot decide the membership in L^A within the allotted time of $O(2^n/\log n)$.

For each stage i, define the oracle A in such a way that the i-th machine M_i^A is incorrect about the membership of 1^n in L^A for some n that is large enough to ensure that M_i^A cannot check all possible strings of length n within $O(2^n/\log n)$ steps.

To show that L^A is in NP^A, construct a non-deterministic oracle machine that guesses a string of length n and queries the oracle to check if this string is in A. If the oracle responds affirmatively, the machine accepts; otherwise, it rejects. This machine decides L^A in non-deterministic polynomial time with access to A.

By construction, no machine in $\mathsf{TIME}^A(2^n/\log n)$ can decide L^A because for each machine M_i^A, we have chosen n during the diagonalization process to ensure that M_i^A cannot decide the membership of 1^n in L^A within the time bounds.

By this construction, we have created an oracle A such that $\mathsf{TIME}^A(2^n/\log n) \neq \mathsf{NP}^A$, since there is a language that is in NP^A but not in $\mathsf{TIME}^A(2^n/\log n)$. This proves that there is an oracle relative to which these complexity classes are different.

7. (5 points) Let $\mathsf{TISP}[t(n), s(n)]/a(n)$ be the class of languages decidable by advice machines which run in time $t(n)$ **and** use no more than $s(n)$ space and are given $a(n)$ advice. For what $f(n)$ is it true that $\mathsf{TISP}[t(n), s(n)]/a(n) \subseteq \mathsf{SIZE}(f(n))$?
It is known that for a machine running in time $f(n)$, there exists a circuit of size at most $f^2(n)$. However, this does not take into account the space complexity $s(n)$ or the advice $a(n)$.

Given that:

(a) The space used by the Turing machine corresponds to the width of the circuit and affects the total number of different configurations the machine's tape can be in, which influences the size of the circuit

(b) The advice string, which can be hardwired into the circuit, increases the size of the circuit by at least $a(n)$ because it has to be encoded within it

(c) The computation time $t(n)$ suggests that the circuit will need to have a depth proportional to $t(n)$, affecting the overall size of the circuit

Combining these considerations, we would expect that a boolean circuit simulating such a machine would need to be large enough to account for all these aspects. Specifically, the size $f(n)$ would have to accommodate:

- The time complexity, by having enough gates to represent each step of the computation ($O(t(n))$)
- The space complexity, by having enough parallel paths to represent all possible space configurations, which is exponential in the space bound ($O(2^{s(n)})$)
- The advice complexity, by including gates to encode the advice string ($O(a(n))$)

A rough estimate for the size of the circuit that would simulate such a Turing machine could be:

$$f(n) = O(t(n) + 2^{s(n)} + a(n))$$

While the exact function $f(n)$ would be determined by a detailed analysis of the machine and the problem it solves, the size $f(n)$ of the circuit is expected to be at least large enough to represent the time complexity, exponentially larger than the space complexity, and inclusive of the advice length.

8. (5 points) Prove that if $SAT \leq_p \overline{SAT}$ then PH collapses to NP.

Assume SAT $\leq_p \overline{\text{SAT}}$. This means there exists a polynomial-time reduction from the satisfiability problem to its complement. Formally, given a SAT instance ϕ, we can construct in polynomial time an instance ψ such that ϕ is satisfiable if and only if ψ is not satisfiable.

If SAT $\leq_p \overline{\text{SAT}}$, then NP $\subseteq$ coNP. This is because any problem in NP, in particular SAT, can be decided by an NP machine using the reduction to $\overline{\text{SAT}}$, and then running the NP algorithm for $\overline{\text{SAT}}$ (which exists because SAT is NP-complete), and finally inverting the answer.

The polynomial hierarchy is defined as a hierarchy of complexity classes where each level alternates between existential and universal quantifiers. The first level is NP, followed by coNP, and so on, with each level defined by Turing machines with oracles for the problems in the level below.

Now, the collapse of PH to NP happens inductively:

- The second level Σ_2^P is defined as NP problems with an oracle for coNP
- If NP = coNP, then an NP machine can simulate a coNP oracle, thus Σ_2^P problems can be solved in NP
- Similarly, the third level Π_2^P, which consists of coNP problems with an oracle for Σ_2^P, collapses to coNP, and therefore to NP, by the same logic
- This inductive argument continues for all levels of PH

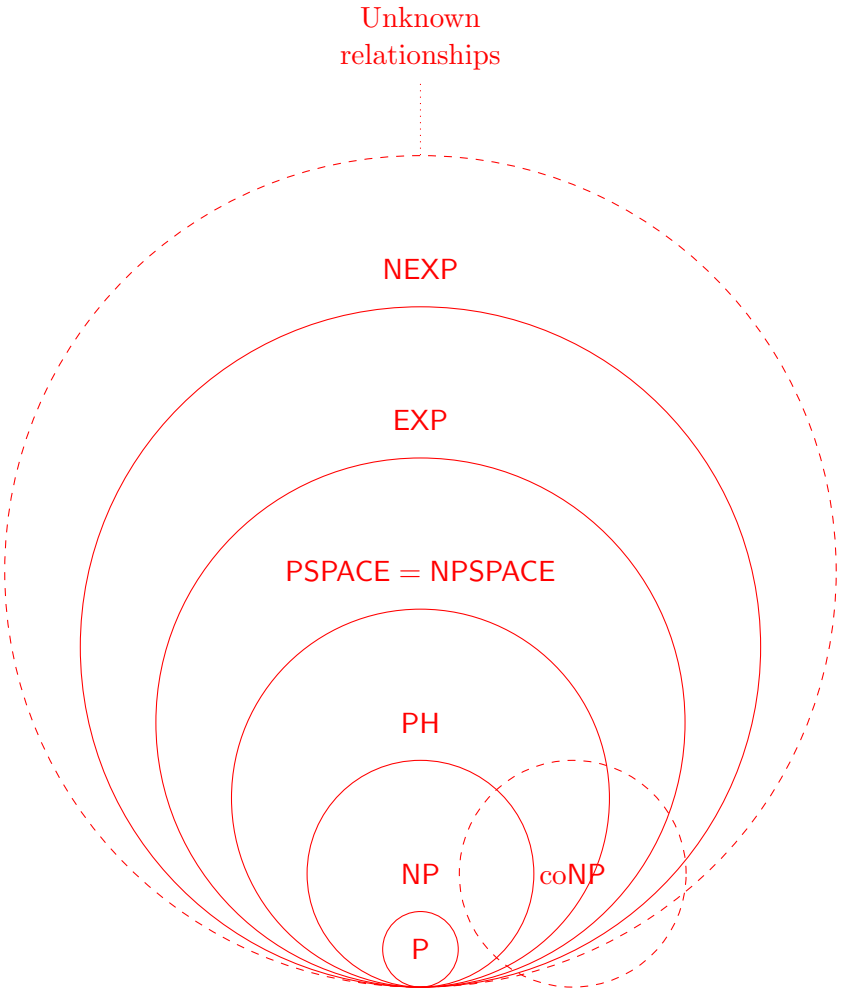
Since all levels of PH can be simulated within NP, we conclude that PH collapses to NP.

Thus, the existence of a polynomial-time reduction from SAT to $\overline{\text{SAT}}$ leads to the conclusion that every level of PH is no more powerful than NP, implying that PH collapses to NP.

9. (5 bonus points) Draw a cool venn or arrow (or other) diagram or representing the relationship between the classes:

   P, NP, PSPACE, NPSPACE, PH and its levels, EXP, NEXP, and coNP.

   If the containment or equality between two classes is unknown, denote it, perhaps with a different color or dotted line. The best diagram will receive two more bonus points.



In case the picture isn't detailed enough:

- (P $\subseteq$ NP)
- (NP ?= coNP)
- (NP $\cup$ coNP $\subseteq$ PH)
- (PH $\subseteq$ PSPACE)
- (NPSPACE = PSPACE)
- (PSPACE $\subseteq$ EXP)
- (EXP $\subseteq$ NEXP)