HW6

● Graded

Student

Vidit Dharmendra Pokharna

Total Points

39 / 40 pts

Question 1

Q1

8 / 8 pts

✓ **– 0 pts** Correct

– 2 pts Checks for string in correct form but does not check for satisfiability or vice versa

– 1 pt Minor Error

– 4 pts Major Error

Question 2

Q2

8 / 8 pts

✓ **– 0 pts** Correct

– 1 pt Minor error

– 2 pts Not correctly calculating complexity according to size of ϕ (Must give upper bound as $2^{O(log(n))}$ or similar)

– 4 pts Use the decider of L as the decider of SAT

– 5 pts Saying the padding part is super-polynomial (forgot it's also part of input for L)

– 5 pts Not built a decider for SAT with L

Question 3

Q3

8 / 8 pts

✓ **– 0 pts** Correct

– 1 pt Minor Error

– 4 pts Major Error

Question 4

Q4

2 / 2 pts

✓ **– 0 pts** Correct

– 2 pts Incorrect

– 1 pt Partially correct

– 1 pt Proof missing

Question 5

Q5

1 / 2 pts

– 0 pts Correct

– 0.5 pts Mostly accurate explanation

✓ **– 1 pt** Basic understanding but has significant inaccuracies

– 2 pts Incorrect/No attempt made

Question 6

Q6

4 / 4 pts

✓ **– 0 pts** Correct

– 2 pts Major Error

– 1 pt Minor Error

– 4 pts Incorrect/No Answer

Question 7

Q7

8 / 8 pts

✓ **– 0 pts** Correct

– 2 pts Missing O(f(n)) argument

– 6 pts Major Error

– 2 pts Minor Error

– 8 pts Missing

Question assigned to the following page: 1

CS 4510 Automata and Complexity 3/28/2024

Homework 6: Complexity (easy mode)

Vidit D. Pokharna *Due: 4/11/2024*

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.

We proved Ladner's theorem with a quasi-polynomial sized padding of $2^{\sqrt{n}}$. In this assignment, you will reprove Ladner's Theorem with a different quasipolynomial padding of $2^{(\log(n)^3)}$. Assume the Exponential Time Hypothesis, that there is no sub exponential time algorithm for SAT. It cannot be solved by any $2^{o(n)}$ time algorithm. Consider the language:

$$L = \{\langle \phi, 1^{2^{(\log(|\phi|)^3)}} \rangle \mid \phi \in SAT\}$$

Note that the exponent of the padding is **not** of length $\log(\log(\log(|\phi|)))$, but of $(\log|\phi|)^3$

1. (8 points) Prove that $L \in \mathsf{NP}$

   Consider a verifier V that takes as input a tuple $\langle w, c \rangle$ and a certificate s, and performs the following steps:

   - Verify that w is of the form $\langle \phi, 1^{2^{(\log(|\phi|)^3)}} \rangle$. This involves checking if the second component of w is a string of 1s whose length is $2^{(\log(|\phi|)^3)}$. This can be done by taking the logarithm of the length of ϕ, cubing the result, and then exponentiating 2 to the power of this number, which all are polynomial-time operations.
   - Use the certificate s as a proposed satisfying assignment for ϕ. Since SAT is in NP, there exists a polynomial-time verifier that can check if s indeed satisfies ϕ.

   If both steps are successful, V accepts, proving that w is in L. The verifier operates in polynomial time because both checking the padding length and verifying a satisfying assignment for a Boolean formula can be done in polynomial time. Hence, $L \in \mathsf{NP}$.

Question assigned to the following page: 2

2. (8 points) Prove that $L \notin \mathsf{P}$

Assume to the contrary that L is in P, which means there exists a deterministic polynomial-time algorithm A that decides L.

Given an input ϕ to SAT, we construct a new input $y = \langle \phi, 1^{2^{(\log(|\phi|)^3)}} \rangle$ and pass it to algorithm A. Since A is polynomial in the size of its input, and since the size of y is dominated by the exponential padding $1^{2^{(\log(|\phi|)^3)}}$, the running time of A on input y would be polynomial in $2^{(\log(|\phi|)^3)}$.

Now, the running time of A is some polynomial of its input size, say $O((n+2^{(\log(n)^3)})^k)$, where n is the size of ϕ and k is some constant. For input y, this would be $O((2^{(\log(n)^3)})^k)$, which is sub-exponential with respect to the original size of ϕ, as $2^{(\log(n)^3)}$ is quasipolynomial in n.

If A can decide L in polynomial time with respect to this quasipolynomial padding, it implies that we can decide SAT in sub-exponential time with respect to the size of ϕ, as we'd just be running A on the padded input. However, this contradicts the Exponential Time Hypothesis, which asserts there are no sub-exponential time algorithms for SAT.

Hence, we conclude that $L \notin \mathsf{P}$.

Question assigned to the following page: 3

3. (8 points) Prove that L is not NP-complete.

To argue that L is not NP-complete, we can say that an NP-completeness assertion for L would imply the existence of a polynomial-time reduction from SAT to L, which is in conflict with the Exponential Time Hypothesis (ETH). Let's consider the implications if L were NP-complete. There would then be a polynomial-time reduction function f that converts any instance ψ of SAT into an instance $\langle \phi, 1^{2^{(\log(|\phi|)^3)}} \rangle$ of L, where ϕ is some Boolean formula potentially different from ψ.

Given that f operates in polynomial time, there's a constant k such that the size of the output from f, $|f(\psi)|$, is upper-bounded by the size of the input $|\psi|$ raised to the power of k. This reflects the principle that the output of polynomial-time reductions is polynomially related to the size of their input.

Considering the requirement for polynomial-sized outputs from reductions, it follows that if $|f(\psi)|$ is polynomial in $|\psi|$, then ϕ must be sufficiently short so that $2^{(\log(|\phi|)^3)}$ is also polynomial in $|\psi|$. More explicitly, $|f(\psi)| = n^k$ for $n = |\psi|$ implies $|f(\psi)| \geq 2^{(\log(|\phi|)^3)}$, resulting in $|\phi|$ being at most $O(n^{1/3})$ or $O((\log n)^3)$.

Given this significant size difference, $|\phi| \ll |\psi|$, checking all possible assignments for ϕ becomes easier than for ψ. For any instance ψ of SAT, by applying f, we transform it into $\langle \phi, 1^{2^{(\log(|\phi|)^3)}} \rangle$ and then proceed to verify every possible assignment for ϕ. The computational effort for this verification is predominantly determined by the number of assignments to ϕ, which would be $2^{O((\log n)^{1/3})}$, not $2^{o(n)}$. Such a result defies the ETH.

Hence, given the constraints imposed by the ETH, we must conclude that L cannot be NP-complete.

Question assigned to the following page: 4

4. (2 points) Discuss what would happen to L had the padding been polynomial instead, say $\langle \phi, 1^{n^3} \rangle$.

   If the language L had been defined with a polynomial padding, specifically $\langle \phi, 1^{n^3} \rangle$, L would remain within NP. This is because polynomial padding doesn't increase the complexity of verifying whether a provided certificate (a solution for ϕ) is correct; in fact, it's still a process that can be completed in polynomial time with respect to the input size.

   However, the polynomial padding changes the nature of L in relation to the NP-completeness discussion. With padding of size n^3, L would not be considered NP-intermediate under the assumption that NP-intermediate languages exist (as suggested by Ladner's theorem), since the padding size n^3 falls within the polynomial bounds and wouldn't lead to a violation of the Exponential Time Hypothesis (ETH).

   Therefore, the hypothetical polynomial-time reduction function f, which transforms a SAT instance ψ into an instance of L as $\langle \phi, 1^{n^3} \rangle$, would not serve as evidence against ETH, as it would not imply the existence of a sub-exponential time algorithm for SAT. Thus, it wouldn't be feasible to demonstrate that L is NP-complete solely on the basis of the Exponential Time Hypothesis if the padding is polynomial.

Question assigned to the following page: 5

5. (2 points) Discuss what would happen to L had the padding been exponential instead, say $\langle \phi, 1^{2^n} \rangle$.

   If the padding within language L was altered to be exponential, for instance $\langle \phi, 1^{2^n} \rangle$, we would see a drastic impact on the computational characteristics of L. While the process of verifying a candidate solution for ϕ would not be hindered by the length of the padding and could still be done in polynomial time, thus keeping L in NP, the practical implications of such padding would be significant.

   The exponential padding would make instances of L extremely large, which would greatly affect the storage and handling of these instances, making them largely impractical for computation. Additionally, if any polynomial-time reduction from a different problem to L were to be attempted, it would inherently lead to an exponentially larger instance size due to the padding. This exponential growth could impede the ability to find reductions that are feasible in polynomial time, thereby complicating the verification of whether L is NP-complete. The properties that define NP-completeness require that problems must not only be verifiable in polynomial time on a non-deterministic Turing machine but also be reducible from any other NP problem in polynomial time, which may not hold true with such exponential growth in instance size.

Question assigned to the following page: 6

6. (4 points) Prove that for $f(n) = 2^{((\log n)^{1+\varepsilon})}$ where $\varepsilon > 0$ any positive real number that $f(n)$ is greater than every polynomial and less than every exponential.

   For the polynomial comparison, consider the limit of the ratio of $f(n)$ to a polynomial n^k:

$$\lim_{n\to\infty} \frac{f(n)}{n^k} = \lim_{n\to\infty} \frac{2^{(\log n)^{1+\varepsilon}}}{n^k}.$$

   As n becomes large, $(\log n)^{1+\varepsilon}$ increases faster than $k \cdot \log n$ (which is the exponent in n^k when expressed as $2^{k \log n}$). Thus, the limit is infinite, meaning $f(n)$ outgrows any polynomial n^k.

   For the exponential comparison, consider the limit of the ratio of $f(n)$ to an exponential function 2^{cn}:

$$\lim_{n\to\infty} \frac{f(n)}{2^{cn}} = \lim_{n\to\infty} \frac{2^{(\log n)^{1+\varepsilon}}}{2^{cn}}.$$

   In this case, $(\log n)^{1+\varepsilon}$ grows slower than cn as n becomes large, and so the limit is zero. This demonstrates that $f(n)$ grows slower than any exponential function 2^{cn}.

   Having shown that $f(n)$ exceeds any polynomial growth rate yet remains beneath any exponential growth rate, we can conclude that $f(n) = 2^{(\log n)^{1+\varepsilon}}$ indeed qualifies as a quasi-polynomial function.

7. (8 points) Let M be an $f(n)$ space bounded machine and consider its computation on some word w. Prove that there exists a CNF formula ϕ of size $O(f(n))$ such that for any strings C_i, C_{i+1}, we have that $\phi(C_i, C_{i+1}) = 1$ if and only if C_i, C_{i+1} are configurations of M on w and $C_i \vdash C_{i+1}$. There are two solutions to this from two textbooks which are incomplete, see the attached screenshots. Your job is to fill in the details.

A configuration C of M can be represented by the state of the machine, the contents of the tape up to $f(n)$, and the position of the tape head. We'll describe a CNF formula that checks both the validity of a configuration and the correctness of the transition from one configuration to the next.

For a given configuration C, we must encode into ϕ the content of each tape cell, the state of the machine, and the head position. This encoding can use $O(f(n))$ variables because there are $f(n)$ cells, a constant number of machine states, and $f(n)$ possible head positions.

For each tape cell i and tape alphabet symbol s, introduce a variable $x_{i,s}$ which is true if and only if cell i contains symbol s. Ensuring that each cell contains exactly one symbol can be achieved by a conjunction of clauses that, for each cell i, include a disjunction $\bigvee_{s\in\Sigma} x_{i,s}$ asserting that cell i contains at least one symbol and a series of clauses $\neg x_{i,s} \vee \neg x_{i,t}$ for all $s \neq t$ to ensure it doesn't contain two different symbols.

To check that configuration C_{i+1} legally follows from C_i, we encode the transition rules of M. This involves verifying that the machine's state and tape contents, along with the position of the head, evolve according to M's rules.

For every possible transition of M, we add clauses that capture the condition under which the transition is valid. For example, if the machine reads symbol a in state q and the transition dictates writing symbol b, moving to the right, and switching to state q', we add clauses that enforce this. The transition requires not only checking the current state and tape symbol but also ensuring that the next state and tape symbol are set accordingly and that the head moves as specified by the transition function.

The CNF formula ϕ is thus composed of two parts:

- ϕ_{cell}: This ensures every cell has one and only one symbol
- ϕ_{move}: This ensures the legality of the transition from C_i to C_{i+1}

Since the number of clauses needed to represent each cell's content and each transition is constant and independent of n, and since we only need to encode $f(n)$ cells for a space-bounded machine, the total number of clauses is proportional to $f(n)$, keeping the size of ϕ within $O(f(n))$.

Finally, to ensure that ϕ captures the entire computation of M on w, we would take the conjunction of the formulas for each pair of successive configurations along the computation path. If M accepts w, there exists a sequence of configurations leading from the initial to the accepting configuration such that ϕ evaluates to 1 for each consecutive pair, proving that such a ϕ exists.