COMS 4236: Intro to Computational Complexity — Released: October 7, 2026

Homework 2 Solutions

Instructor: *Toniann Pitassi* — Due: October 15, 2026

1 Question 1

Solution.

(a) If L has a verifier that satisfies these requirements, we give a logspace NTM for L. When supplied with the input tape x, the NTM runs the verifier on the input tape, nondeterministically choosing the bit at the head position of the certificate tape. It only requires constant space to store the bit at the head position of the certificate and we can forget the bit after the certificate head moves because we know that it will not go back. All other computation used to simulate the verifier takes place in logspace by our assumption that the verifier is logspace.

Now suppose we have a logspace NTM for L. Our verifier will read a certificate consisting of the nondeterministic choices and simulate the computation path of the NTM. All configurations of the NTM are logarithmic sized, so there are polynomially many possible configurations. Removing loops, any computation path has at most polynomially-many steps. This means that the verifier will run in polytime, and also that there are polynomially-many nondeterministic choices to store in the certificate.

(b) Since a verifier of this type is an NP verifier, these languages are a subset of NP. Now, we show that every language in NP also has a verifier of this type. Suppose L has an NTM. As the certificate, take the transcript of all configurations on the accepting computation path. Since each configuration has polynomial size, and there are polynomially-many steps, the length of this certificate is polynomial.

Given this certificate, a verifier will check that the computation path begins in the start state, proceeds according to the transition relation, and ends in the accepting state. It stores counters marking the head positions and the step. To check the computation path, it checks that the unchanged information is the same, and the changed information corresponds to the transition relation. This is polynomially-many checks, requiring logarithmic-sized indices into the certificate. There are polynomially-many steps, so the total check is polytime and fits in log space.

2 Question 2

Solution.

By the space hierarchy theorem, there exists $L \in \mathrm{DSPACE}(n) \setminus \mathrm{DSPACE}(n/\log n)$. Suppose A has an algorithm that runs in time $n^{2-\varepsilon}$. Let $T = n^2/\log^4 n$. $n^{2-\varepsilon} < T$ for sufficiently large n, so $A \in \mathrm{DTIME}(T)$. By Williams, this would imply that T runs in space $\sqrt{T \log T} = \Theta(n/\log^{3/2} n)$. But

this contradicts the fact that $A \notin \text{DSPACE}(n/\log n)$.

3 Question 3

Solution.

(a) Let the original contents of the registers be (R_x, R_y, R_o). After the first row, the registers are $(R_x + x, R_y, R_o + R_x \odot R_y)$. After the second row, they are $(R_x + x, R_y + y, R_o + R_x \odot R_y + (R_x + x) \odot R_y)$, which simplifies to $(R_x + x, R_y + y, R_o + x \odot R_y)$. After the third row, the registers are $(R_x, R_y + y, R_o + x \odot R_y + (R_x + x) \odot (R_y + y))$, which simplifies to $(R_x, R_y + y, R_o + R_x \odot (R_y + y) + x \odot y)$. After the fourth row, the registers are $(R_x, R_y, R_o + R_x \odot (R_y + y) + x \odot y + (R_y + y) \odot R_x)$, which simplifies to $(R_x, R_y, R_o + x \odot y)$.

(b) To build a program for OR, observe $x \vee y = x + y + x \odot y$. We can run the child programs that add their values to chosen registers to obtain the x and y, and we can use the procedure from part (a) to obtain the $x \odot y$.

We don't need fresh registers because the operations preserve the values of the other registers.

The reason we are not done proving Cook-Mertz is that a uniform evaluator also has to keep track of node, program position, and recursion/control information (not just evaluating the functions at every node).

4 Question 4

Solution.

(a) Our algorithm will be to draw $z_i \sim \{0,1\}^v$ independently and uniformly at random for $i = 1, \ldots, k$. Then, output $\widehat{p} := \frac{1}{k} \sum_{i=1}^{k} \varphi(v_i)$.

Applying Chernoff to $X_i := \varphi(v_i)$ for $i = 1, \ldots, k$, we find $\Pr[k|\widehat{p} - p| \geq a] \leq 2e^{-2a^2/k}$. Substituting $a = k\varepsilon$, we get $\Pr[|\widehat{p} - p| \geq \varepsilon] \leq 2e^{-2k\varepsilon^2}$. Setting $k = \ln(2/\delta)/(2\varepsilon^2)$ suffices to achieve the desired bound. This is $O(\varepsilon^{-2} \ln(1/\delta))$.

(b) We might hope that we can run this randomized algorithm, and output 1 if a formula has acceptance probability greater than 0. The issue comes when trying to detect the difference between 0 and 1 satisfying assignments.

Suppose φ has one satisfying assignment. Then, $p = 2^{-v}$. To observe an empirical probability greater than 2^{-v-1}, we need $\varepsilon \leq 2^{-v-1}$. Thus, to achieve δ error, we need $k = O(2^{2v} \ln(1/\delta))$ trials. For constant δ, this is exponentially many.

In fact, the probability that we draw unsatisfying assignments in all k rounds is $(1 - 2^{-v})^k \leq e^{-2^{-v}k}$. If we want this probability to be constant, we require $k = \Theta(2^v)$.

Note that we have not shown that SAT is not in BPP, it only shows that the naive sampling algorithm does not place SAT in BPP.