

Due on September 29

In class, at the beginning of the lecture

The work you submit must be your own. You may discuss problems with each other; however, you should prepare written solutions alone. In particular, you should not leave with any written notes from such discussions. The style and clarity of your answers will be an important factor in the grade.

Each question is worth 25%.

1. (a) If t is a nae-assignment, then in each clause $t(l_1) = T, t(l_2) = F$, for some l_1, l_2 , so now $\bar{t}(l_1) = F, \bar{t}(l_2) = T$.
 - (b) $c_i = (l_1 \vee l_2 \vee l_3) \xrightarrow{f} c'_i = (l_1 \vee l_2 \vee z_i) \wedge (\bar{z}_i \vee l_3 \vee b)$ where z_i is a new variable for each clause c_i , and b is a single new variable for all the clauses. The result of the (log-space) mapping f is the conjunction of two clauses c'_i . Correctness: suppose t satisfies $\bigwedge_i c_i$. Let t' be defined as follows $t'(b) = F$, and if $t(l_1 \vee l_2) = T$, then $t'(\bar{z}_i) = T$, and if $t(l_1 \vee l_2) = F$, and $t(l_3) = T$, then $t'(z_i) = T$. Clearly, t' is a nae-assignment. Now we do a similar argument for the other direction.
2. The trick is to map an instance of NEASAT to an instance of DEG2POLY as follows: for each variable x_i , create $x_i^2 = 1$ (so variables can take on the values ± 1), and map each clause $c = (l_1 \vee l_2 \vee l_3)$ to $(m(l_1) + m(l_2) + m(l_3))^2 = 1$ (where $m(l) = (1 - x)$ if $l = x$ and $m(l) = x$ if $l = \bar{x}$). Correctness: if t is a nae-assignment satisfying the formula, then t' , defined as $t'(x_i) = +1$ if $t(x_i) = T$ and $t'(x_i) = -1$ if $t(x_i) = F$, satisfies the equations since $(m(l_1) + m(l_2) + m(l_3)) = \pm 1$. Similar argument for the other direction.
3. Following the hint, we give a polytime algorithm for SAT: it works in stages: at stage 0, let $C = \{\alpha\}$, where f is the input formula. At stage $(i + 1)$, $C = \{\alpha_1, \dots, \alpha_n\}$, and we create $C' = \{\alpha_1[v_{i+1} = T], \dots, \alpha_n[v_{i+1} = F], \alpha_1[v_{i+1} = T], \dots, \alpha_n[v_{i+1} = F]\}$. We now prune C' as follows: we compute $g(\alpha_j[v_{i+1} = \text{TruthValue}])$ for every $j \in [n]$ and every $\text{TruthValue} \in \{T, F\}$. Whenever we get two formulas that map to the same string in $\{1\}^*$, we keep only one of them. We let C be the result of this pruning. At the end, when no variables are left, we check if at least one formula in C is true. Why is this algorithm polytime? In fact, why does it work?
4. It is enough to show that if $\mathbf{NP} \subseteq \mathbf{P/poly}$ then $\Pi_2^p \subseteq \Sigma_2^p$. Consider $(\forall y \leq p(|x|))(\exists z \leq p(|x|))\alpha(x, y, z)$. Since $(\exists z \leq p(|x|))\alpha(x, y, z)$ is an \mathbf{NP} predicate, it can be replaced by a SAT predicate, i.e., by $(\forall y \leq p(|x|))(\exists z \leq p(|x|))\text{SAT}(F(x, y), z)$, where $F(x, y)$ is a polytime function whose output is a boolean formula, and $\text{SAT}(F(x, y), z)$ means that z satisfies $F(x, y)$. Since $\mathbf{NP} \subseteq \mathbf{P/poly}$, there are polysize circuits for satisfiability. Using the self-reference of SAT, there are in fact polysize circuits for SAT that output the satisfying assignment (what are these circuits?). So now we can say $(\exists \langle C_1, \dots, C_m \rangle \leq q(|x|))(\forall y \leq p(|x|))\text{SAT}(F(x, y), C(F(x, y)))$.