

Due on October 27

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.

1. Such a game can be described with a (fully) quantified boolean formula. Such a formula is either true (in which case the first player has a winning strategy), or false (in which case the second player has a winning strategy).
2. We prove the claim $\Sigma_i \mathbf{P} = \Sigma_i^{\text{Alt}} = \Sigma_i^{\text{P}}$ by induction on i . The basis case is simple, as all three are equal to \mathbf{P} (by definition). To show the inductive step assume the claim for i , and show it for $(i + 1)$.

(a) $\Sigma_{i+1} \mathbf{P} \subseteq \Sigma_{i+1}^{\text{Alt}}$

Suppose $L \in \Sigma_{i+1} \mathbf{P}$. Then L is decided by an \mathbf{NP} TM with a $\Sigma_i \mathbf{P}$ oracle, i.e., M^O . By IH this oracle O is in Σ_i^{Alt} . Consider the ATM which simulates M , except for two enhancements: (i) it first guesses at most polynomially many queries, *together with* the answers to those queries, i.e.,

$$\{(\langle q_1, w_1 \rangle, \text{“yes”}), (q_2, \text{“no”}), \dots, (\langle q_m, w_m \rangle, \text{“yes”})\},$$

and writes it all down. (Note that the “yes” queries also have a polylong string attached to the them; its role will become clear later.) (ii) It now simulates M , but when M is about to query the oracle for the i -th time, it checks that the query is indeed q_i . If this test passes, it responds with the answer it has guessed.

Some branches of this \mathbf{NP} (i.e., Σ_1^{Alt}) computation will accept, and some will reject. On the accepting branches, we must verify that we guessed the correct answers to the queries. We do this as follows: the oracle O is a Σ_i^{Alt} language, so consider \bar{O} , a Π_i^{Alt} language. Branch universally on all the m queries (polynomially many of them), and then, if q_i is a “no” query, check if $q_i \in \bar{O}$ (a Π_i^{Alt}) computation, and if q_i was a “yes” query, check that $\langle q_i, w_i \rangle \in O'$, where O' is just like O , except it also takes as input “guidance” for the initial run of existential non-deterministic choices its Σ_i^{Alt} machine makes, and thus it is *de facto* a Π_{i-1}^{Alt} machine.

Thus, the w_i are *witnesses* of what the machine for O must do on the initial existential run of the “yes” input q_i , in order to eventually accept.

Altogether it is a $\Sigma_{i+1}^{\text{Alt}}$ computation. This shows that $\Sigma_{i+1} \mathbf{P} \subseteq \Sigma_{i+1}^{\text{Alt}}$.

(b) $\Sigma_{i+1}^{\text{Alt}} \subseteq \Sigma_{i+1}^{\text{P}}$

Suppose that L is decided by a $\Sigma_{i+1}^{\text{Alt}}$ ATM M . A computational path of M , on input x , can be described with a sequence of $(i + 1)$ numbers in base d , where

d is the degree of nondeterminism of M . For example, y_1, y_2, y_3 describe a sequence of $|y_1|$ -many existential choices, followed by $|y_2|$ -many universal choices, followed by a block of $|y_3|$ -many existential choices. Let $R(y_1, y_2, \dots, y_{i+1}, x)$ be a predicate which holds whenever the sequence of choices y_1, y_2, \dots, y_{i+1} on input x lands at an accepting leaf. Since M is polytime, R is polytime. Clearly, M accepts x iff $(\exists y_1)(\forall y_2) \cdots (Qy_{i+1})R(y_1, y_2, \dots, y_{i+1}, x)$ (bounds on quantified variables omitted). This finishes the proof.

(c) $\Sigma_{i+1}^p \subseteq \Sigma_{i+1}\mathbf{P}$

Suppose $L \in \Sigma_{i+1}^p$, so $w \in L \iff (\exists x)\alpha(x, w)$, where $\alpha(x, w)$ is a Π_i^p formula, so $\neg\alpha(x, w)$ is a Σ_i^p formula, and so by induction $\neg\alpha(x, w)$ is a $\Sigma_i\mathbf{P}$ oracle. Now decide L with a $\Sigma_{i+1}\mathbf{P}$ machine as follows: non-deterministically guess x_0 (of poly length), and query the $\Sigma_i\mathbf{P}$ oracle for $\neg\alpha(x_0, w)$, and accept iff the answer comes back “no”.

3. Note that H is well defined. This is an issue, because H is defined in terms of itself, i.e., recursively, but note that to compute $H(n)$ we only need to consider the values of $H(k)$ for $k \leq \log n$.

Let M_1, M_2, M_3, \dots be a list of all TMs where each TM occurs infinitely often.

We show first that P-SAT is not in \mathbf{P} . Suppose that it is, and it is decided by some M running in time kn^k . There is an $i > k$ such that $M = M_i$. Therefore, by the definition of p , for all $n > 2^{2^i}$, $p(n) \leq i$. But this means that for $n > 2^{2^i}$, P-SAT is just SAT padded with at most n^i many #s. So if P-SAT were in \mathbf{P} , so would SAT (by the following polytime algorithm: given ϕ , check if $\phi\#^j$, for some $j \in [n^i]$ is in P-SAT), which is not possible unless $\mathbf{P} = \mathbf{NP}$.

We show second that P-SAT is not \mathbf{NP} -complete.

Assume that it is \mathbf{NP} -complete. Then $p(n)$ tends to infinity with n . We show this by showing that for every integer i , there are only finitely many n 's such that $p(n) = i$. Since P-SAT $\notin \mathbf{P}$, for each i we know that there exists an x such that given time $i|x|^i$, M_i gives the incorrect answer to the question $x \stackrel{?}{\in}$ P-SAT. Then, we know (from the definition of p) that for every $n > 2^{|x|}$, $p(n) \neq i$.

If $p(n)$ tends to infinity with n , this means that the padding is of super-polynomial size. Suppose P-SAT is \mathbf{NP} -complete, so $\text{SAT} \leq_L \text{P-SAT}$. This reduction takes ψ , such that $|\psi| = n$, to $\phi\#^{|\phi|^{p(|\phi|)}}$, such that $|\phi\#^{|\phi|^{p(|\phi|)}}|$ is $O(n^k)$ for some fixed k . Therefore, $|\phi| + |\phi|^{p(|\phi|)}$ is $O(n^k)$. But this means that $|\phi|$ must be $o(n)$. Using this fact, we can now design a polytime algorithm for SAT which applies this procedure repeatedly, each time obtaining a smaller ϕ , until it is of constant size, and can be solved by brute force. It follows that $\mathbf{P} = \mathbf{NP}$, contradiction.

4. First note that there must be an m such that $T_m^n = T_{m+1}$. Then, T_m^n will contain all the strings of length at most n derivable from S . Suppose we know $t = |T_m^n|$. Then, we can check if a given w is *not* in the language as follows: for each $w \in (\Sigma \cup V)^{\leq n}$, guess a derivation (step by step, so it fits in linear space), and if we succeed, decrease t by 1.

If we ever guess a derivation for w , reject. If at the end $t > 0$, also reject (we missed some x which had a derivation, and it might have been w). If $t = 0$, accept w .

It remains to show how to compute $|T_m^n|$. First, note that $T_0^n = \{S\}$, so $|T_0^n| = 1$. We now use the inductive counting method to compute $|T_{i+1}^n|$ from $|T_i^n|$ in non-deterministic linear space: for each $x \in (\Sigma \cup V)^{\leq n}$, to see if $x \in T_{i+1}^n$, go through all $y \in (\Sigma \cup V)^{\leq n}$, and check if $y \in T_i^n$ (by guessing a derivation of length at most i) and $y \Rightarrow x$. Since we know $|T_i^n|$, we will know if we missed some y .

We stop when we find an m such that $|T_m^n| = |T_{m+1}^n|$. But how do we know that this m is not very big (perhaps it needs more than $O(n)$ many bits to be encoded)? We know that we only need consider derivation which are no longer than the number of possible sentential forms of length at most n , and there are at most $|\Sigma \cup V|^{n+1}$ -many sentential forms of length at most n , and this number can be encoded with $O(n)$ many bits.

5. First, assume once again that we have an enumeration M_1, M_2, M_3, \dots of all nondeterministic TMs such that each TM M appears infinitely often. Thus, given any M , and given any i_0 , we can always find an $i \geq i_0$ such that $M = M_i$.

Following the hint, we define the diagonal machine D as follows: on input $x = 1^n$ (if $x \notin \{1\}^*$, reject outright), compute an i such that $f(i) < n \leq f(i+1)$ (note that this must, and can, be done in time $O(n^{1.5})$). The machine D now considers two cases.

Case 1: $f(i) < n < f(i+1)$ (i.e., n is strictly in between). Then, D simulates M_i on input 1^{n+1} , using non-determinism, in time $n^{1.1}$ (if M_i does not halt within this time, D simply accepts) and outputs the same answer.

Case 2: $n = f(i+1)$, then D accepts 1^n iff M_i rejects $1^{f(i)+1}$ in $(f(i)+1)^{1.1}$ time. For D to know that M_i rejects, it must go through $2^{(f(i)+1)^{1.1}}$ -many branches of M_i on $1^{f(i)+1}$. But this is doable, since the input size is $n = f(i+1) = 2^{f(i)^{1.2}}$.

Thus D is a non-deterministic machine that runs in time $O(n^{1.5})$. We want to show now that $L(D) \notin \text{NTIME}(n)$. Suppose that it is, so we can find an i large enough so that M_i decides $L(D)$ in time $O(n)$ and on inputs of length $n \geq f(i)$, M_i can be simulated in less than $n^{1.1}$ many steps.

We know that for all $f(i) < n < f(i+1)$, $D(1^n) = M_i(1^{n+1})$, and we also know that $D(1^{f(i+1)}) \neq M_i(1^{f(i)+1})$. Together with the fact that $D(x) = M_i(x)$ for all x , this is an untenable situation. (If you have trouble seeing a contradiction, do a drawing.)