University of California, Los Angeles
CS 281 Computability and Complexity

Boran Erol

# Final Exam

## Problem 1

The following is a failed attempt.
The idea is to diagonalize over all possible reductions to create a decidable language $L'$ that can't be reduced to $L$, our hypothetical D-Complete language.
By contradiction, let $L$ be a D-Complete language. Notice that $L$ needs to have infinitely many strings in it and infinitely many strings not in it, otherwise, P = EXP = DOUBLE-EXP and nothing makes any sense since we can hard-code $L$ into our Turing Machines.
We construct a decidable language $L'$ that can't be reduced to $L$ as follows.
Let $f_i$ be an enumeration of all possible polynomial time reductions.
For $i = 0, 1, 2, ...$:
Find some $x, y$ such that $f_i(x) \in L$ and $f_i(y) \notin L$ that we haven't assigned so far. We can do this since $L$ is decidable and contains infinitely many such $f_i(x)$ and $f_i(y)$.
**This is where my solution completely breaks. What if $f_i$ is not injective? It's not true that L contains infinitely many $f_i(x)$ and $f_i(y)$.**
If this were to work, however, I could let $x \notin L'$ and $y \in L'$. $L'$ would be decidable using the reductions but wouldn't be reducible to $L$ by construction.
I also have a question: does this relate to Godel's Incompleteness Theorem, or am I hallucinating?

# Problem 2

Throughout this problem, $n = |x|$.

LEMMA 1. *$EXP \neq NEXP \implies P \neq NP$.*

*Proof.* Let $L$ be a language that's in NEXP but not in EXP. We'll construct a new language $L'$ that is in NP but not in P.
Since $L \in$ NEXP, there exists a deterministic verifier $V$ such that $\forall x \in L : \exists w : V(x, w) = 1$ and $\forall x \notin L : \forall w : V(x, w) = 0$. V runs in time $2^{n^c}$ for some $c > 0$.
Define the new language as follows: $x \in L$ if and only if $x0^{2^n - n} \in L'$.
Notice that we can use $V$ as a verifier for $L'$ and it will runin time $n^c$ since our inputs have exponential length. Therefore, $L'$ is in NP.
Let's now prove that $L'$ is not in P. Assume it is by contradiction. Then, we can convert this algorithm to an exponential time algorithm for $L$ (if the algorithm tries to read the input tape past $x$, we can just read 0's). This would contradict the fact that $L \notin EXP$. □

## Problem 3

LEMMA 2. *If the polynomial hierarchy does not collapse, $PH \subsetneq EXP$.*

*Proof.* Recall from lecture that $PSPACE \subseteq EXP$. Thus, it suffices to show that $PH \subsetneq PSPACE$. Recall that $PH \subset PSPACE$. If $PH = PSPACE$, then TQBF is $PH$-Complete, which implies that the polynomial hierarchy collapses since the existence of a PH-Complete problem implies the collapse the polynomial hierarchy. □

## Problem 4

One solution I had in mind was to let the oracle $O$ be the prover in a PCP proof system and use the verifier $V$ with polynomially many random bits and polynomially many queries. I know this produces NEXP, whereas a determinsitic verifier would produce NP. However, we didn't cover PCP = NEXP in class, so this solution would be too long to write. I couldn't come up with any other solutions.

# Problem 5

Since $BPP \subseteq RP$, it suffices to show that $L \in RP$ if $L \in BPP$. Instead, we'll prove the following lemma, which proves the question since $L \in NP$ and $L \in BPP$ implies $NP \subseteq BPP$ by the completeness of $L$.

LEMMA 3. *If* $NP \subseteq BPP, NP = RP$.

Before I do the proof, here's an overview and intuition behind the proof. Let $M$ be the BPTM that decides SAT.
In order accept only when the $\phi$ is satisfiable, we "extract" a candidate witness from $M$ when $M$ accepts. We do this by using an inductive procedure that resembles the decision-to-search reduction (in fact, is almost identical) to extract a candidate assignment. Then, before accepting, we verify the candidate assignment.

*Proof.* We'll prove by inducting on the number of variables in $\phi$ that BPTMs that decide $\phi$ can also be used to produce a assignment.
Using this, we can just verify whether the candidate assignment produced by the BPTM is actually correct and accept only if it is. This produces an algorithm with only one-sided error, so $SAT \in RP \implies NP \subseteq RP$.
Notice that the base case is trivial: we don't even need to use the BPTM.
Let $M$ be a Turing Machine such that $M(x) = SAT(x)$ with probability greater than $7/8$, and assume we can use $M$ to extract a candidate for CNFs with at most $n$ variables with probability $7/8$ or reject. Let $\phi$ be a CNF with variables $x_1, x_2, ..., x_n, x_{n+1}$. First, run $M(\phi)$ to see if $M$ accepts. If $M$ rejects, reject. If $M$ accepts, let $x_{n+1}$ be 0 or 1 and run $M$ with the modified CNF. If $M$ accepts for either iteration, we can extract a canditate assignment by using the extracted assignments (using induction) for $x_1, ..., x_n$ and letting $x_{n+1}$ accordingly. We can then check whether these assignments actually satisfy $\phi$ and only output accept if it does so.
Let's now analyze this construction. Clearly, we reject with probability at least $7/8$ when $\phi$ is unsatisfiable. Notice that when $\phi$ is satisfiable, we produce a satisfying assignment with probability $49/64$ (since we call $M$ twice). ∎

## Problem 6

Having access to a coin with bias $p$ allows us to approximate $p$ to arbitrary precision by sampling the coin a large number of times (triple exponential time if needed) and calculating the fraction of inputs where the coin evaluated to 1. This can be formalized using Chernoff bounds: the precise analysis isn't significant since we can pad using triple exponential bits, which gives us all the time in the universe and more. (In the Chernoff bound formula covered in class, we can make $n$ as large as we want by using padding.)

Let $PM_i = \langle M_i \rangle 0^{2^{2^{2^k}}}$, given by the lexicographical ordering of $\langle M_i \rangle$. Notice that $PM_i$ undecidable since HALT is undecidable.
Let $p = p_1 p_2 ...$, where $p_i = 1$ if $M_i$ halts on all inputs. Then, given a Turing Machine $M_i$, we can find whether it halts with negligible error probability by sampling $p$ enough times and reading off $p_i$ from the approximation of $p$.

# Problem 7

Let $x \in \{0,1\}^n$ be Alice's string and $y \in \{0,1\}^n$ be Bob's string.
Alice and Bob will interpret $x, y$ as degree-n polynomials in $F_p$ for some $p$ with $4n \leq p \leq 8n$.
Alice can find a such $p$ with probability greater than $2/3$ by sampling enough random numbers and using the AKS algorithm for deterministic primality testing. (If Alice fails to find a prime, she aborts the protocol, which happens with negligible probability.)
Then, Alice will sample a random element of $a \in F_p$ and send it to Bob. They'll both evaluate their polynomials and share the answers. All of this requires logarithmically many bits. Using the Schwarz-Zippel lemma, if Alice and Bob's strings are not equal (the difference of their polynomials is non-zero), they'll catch the error with probability greater than

$$\frac{n}{8n} = \frac{1}{8}$$

Repeating this protocol constantly many times, they can boost their probabilities while still using $O(\log n)$ bits.
This was super fun!

## Problem 8

My idea is as follows: I'll make sure that there's at least some order to the input given to the circuit. More formally, I'll "sort" the $i$th bit of the input and the $n - i$th bit of the input by taking the AND of the two bits and redirecting it to the $i$th bit and taking the OR of the two bits and redirecting it to the $n - i$th bit. Then, if there's a majority, all the bits in the right half of the string will be 1, so I can just AND them and output the result.
Since I'm looking at every bit of the string and propagating them to the right half if they are 1, I will catch every majority. The fact that this circuit has no false positives is obvious.

# Problem 9

This is a consequence of a basic lemma in coding theory and the fact that the Hadamard code has distance $k/2$ for messages of length $k$, which was proved in class. Here's the lemma from coding theory:

LEMMA 4. *Given a code $C$, the following are equivalent:*

1. *$C$ has distance $d \geq 2$*
2. *$C$ can decode at most $\lfloor \frac{d-1}{2} \rfloor$ errors.*

I'll only prove that the second condition implies the second one. This proves that it's impossible to recover codewords where more than 25% of the bits are corrupted, since that would imply that the distance of the Hadamard code is greater than $k/2$, which is a contradiction.

*Proof.* We prove the contrapositive. Let $c_1, c_2$ be codewords corresponding to different string and assume $\Delta(c_1, c_2) \leq d$. Then, notice that the received transmission for both of these codewords can be identical by flipping at most $d/2$ bits for both of them. Then, no decoding function can possibly tell them apart. More formally, whatever the decoding function maps to, it makes a mistake. □

**I knew this lemma beforehand, but I used Essential Coding Theory to type it up properly.

## Problem 10

I couldn't solve it. By bounding the length of the proof and guessing the proof, I can get $SAT \in NTIME(n^\epsilon)$ for all $\epsilon > 0$. I couldn't upgrade this to deterministic time. I can't loop over all the proofs, since that makes it $2^{n^\epsilon}$ for all $\epsilon > 0$.

LEMMA 5. *If SAT has a PCP verifier with $o(\log n)$ random bits and $O(1)$ queries, $P = NP$.*

*Proof.* □

## Problem 11

Recall that in the proof of $IP = PSPACE$, we showed the existence of an interactive protocol for TQBF with completeness 1. Since TQBF is PSPACE-Complete, we can reduce every problem in PSPACE to TQBF. Then, we'll have an interactive proof system with completeness 1.
Suppose we increase the soundness parameter to 1. Then, $V$ can only accept if $x \in L$. In other words, a transcript where $V$ accepts can be seen as a witness $w$. We can then construct a verifier $S$ that just checks that this transcript (which includes $V$'s randomness) is a valid transcript. This can be done in polynomial time since $V$ is a polynomial time Turing Machine.

## Problem 12

I am absolutely clueless. There's a $O(n^2)$ trivial algorithm. I've got no clue how to use randomness to improve this. I tried modifying polynomial identity testing, matrix multiplication, and perfect matching in class to get this, but I couldn't. I got most excited about perfect matching with extra conditions on the edges depending on the ordering of the substrings, but I couldn't get a reasonable modification.