CS 151 Complexity Theory **Spring 2017**

Midterm Solutions

Posted: May 10 *Chris Umans*

1. Consider a language $L \in$ **coNEXP**. On an input of length n, the advice will be an exact count of the number of inputs of length n *not* in the language. This is a number between 0 and 2^n, which can be represented using $n+1$ bits. Our **NEXP** machine receives this count $c(n)$ and an input x. It will then guess $c(n)$ distinct strings $x^{(i)}$ together with a witness $w^{(i)}$ of length 2^{n^k} for each. It will then verify that each of the $x^{(i)}$ are in $\overline{L}$ using the guessed witnesses. If not, then it will reject. Otherwise, it will accept if and only if x is not equal to $x^{(i)}$ for any i.

   Now, if $x \in L$, then on some computation path, the **NEXP** machine will correctly guess all of the strings in $\overline{L}$, together with correct witnesses for them, and none of these will be equal to x so it will accept. If $x \notin L$, then the only way the machine can guess $c(n)$ distinct strings in $\overline{L}$ is if x is among these strings, so it will always reject.

2. (a) Let A be an minimum-depth circuit computing f, with depth d. We can assume that A is a formula (because we only care about depth) and we can further assume that all of the $\neg$ gates have been pushed down to the leaves. This is not necessary to solve the problem but makes the exposition shorter.

   We prove by induction on d that there is a protocol for f that exchanges at most d bits. If $d = 1$, then f is just a single variable x_i or its negation, and Alice and Bob can simply announce i without any communication. Thus for $d = 1$ we have $C(f) = 0 \leq 1$, so the base case holds.

   Now assume that the last gate of A is an $\vee$ gate, so $f = g \vee h$, where $D(g) \leq d-1$ and $D(h) \leq d-1$. Alice holds an input x for which $f(x) = 0$, so we know that $g(x) = h(x) = 0$. Bob holds an input y for which $f(y) = 1$, so it must be that either $g(y) = 1$ or $h(y) = 1$. In the first round of communication, Bob will send a single bit $b = g(y)$. If $b = 1$, then both parties can continue with the protocol for g (since $g(x) = 0$ and $g(y) = 1$) which by induction entails at most $d-1$ bits of communication. If $b = 0$, then we know that $h(y) = 1$, since as noted either $g(y) = 1$ or $h(y) = 1$. In this case both parties can continue with the protocol for h (since $h(x) = 0$ and $h(y) = 1$) which by induction entails at most $d-1$ bits of communication.

   If the last gate of A is an $\wedge$ gate, then $f = g \wedge h$, where $D(g) \leq d-1$ and $D(h) \leq d-1$. Alice holds an input x for which $f(x) = 0$, so we know that $g(x) = 0$ or $h(x) = 0$, and Bob holds an input y for which $f(y) = 1$, so $g(y) = h(y) = 1$. In the first round of communication, Alice sends a single bit $a = g(x)$. If $a = 0$, then both parties can continue with the protocol for g (since $g(x) = 0$ and $g(y) = 1$) which by induction entails at most $d-1$ bits of communication. If $a = 0$, then we know that $h(x) = 0$, since as noted either $g(x) = 0$ or $h(x) = 0$. In this case both parties can continue with the

protocol for h (since $h(x) = 0$ and $h(y) = 1$) which by induction entails at most $d - 1$ bits of communication.

The total communication required in either case is at most d bits, as required.

(b) Given a protocol for f on X, Y, we prove by induction on the protocol length ℓ that there is a circuit of depth $\ell + 1$ that outputs 0 on $x \in X$ and 1 on $y \in Y$. If $\ell = 0$, then no information is exchanged between Alice and Bob, so the only way they can correctly agree on an index i is if the function f is a single variable x_i or its negation. In this case there is clearly a circuit of depth 1 computing f correctly on all inputs, so the base case holds.

Now suppose we have an optimal protocol for f on X, Y, that entails ℓ bits of communication. Suppose that Alice sends the first bit. As we saw in class, we can partition the set X into two sets X_0 and X_1, which are the sets of strings that cause Alice to send 0 and 1, respectively. By induction there is a depth $\ell - 1$ circuit C_0 for which C_0 outputs 0 on $x \in X_0$ and 1 on $y \in Y$; similarly there is a depth $\ell - 1$ circuit C_1 for which C_1 outputs 0 on $x \in X_1$ and 1 on $y \in Y$. We define circuit C to be $C_0 \wedge C_1$. This circuit has depth ℓ, and it outputs 0 on $x \in (X_0 \cup X_1) = X$, and 1 on $y \in Y$, as required.

If Bob sends the first bit, then we can partition the set Y into two sets Y_0 and Y_1, which are the sets of strings that cause Bob to send 0 and 1, respectively. By induction there is a depth $\ell - 1$ circuit C_0 for which C_0 outputs 0 on $x \in X$ and 1 on $y \in Y_0$; similarly there is a depth $\ell - 1$ circuit C_1 for which C_1 outputs 0 on $x \in X$ and 1 on $y \in Y_1$. We define circuit C to be $C_0 \vee C_1$. This circuit has depth ℓ, and it outputs 0 on $x \in X$, and 1 on $y \in (Y_0 \cup Y_1) = Y$, as required.

We conclude that there is a circuit of depth ℓ computing f correctly on inputs from X and Y, and by taking $X = f^{-1}(0)$ and $Y = f^{-1}(1)$, we obtain the desired result.

3. Let L be a language in $\mathbf{L/poly}$, and let M be a Turing Machine that decides L in $c \log n$ space with advice $A(n)$, where $A : \mathbb{N} \to \{0,1\}^{N=(n^k)}$. For simplicity we assume that M receives the input x and the advice string $A(|x|)$ on a single read-only input tape, and that it has a single work tape. And, we assume that when M is going to accept or reject, it clears its work tape, resets both read-heads to the first tape square, and enters the state q_{accept} or q_{reject} accordingly. We also assume that the input and the advice string are given over the binary alphabet $\Sigma = \{0,1\}$. Let $p(n)$ be a polynomial bound on the running time of M.

Our branching program for inputs of length n will have nodes for every element of $\Sigma^{c \log n} \times Q \times \{1 \dots p(n)\} \times \{1 \dots c \log n\}$. Note that such a quadruple describes a configuration of M: it contains the contents of the work tape, the current state, and the head position on the input tape, as well as the head position on the work tape. The node corresponding to the start configuration is designated *start*, and the two nodes corresponding to the (unique) accept and reject configurations are designated *accept* and *reject*. We label with i each node (other than *accept* and *reject*) corresponding to a configuration of M in which it is reading the i-th symbol from the read-only input tape. We connect the "0" outgoing edge to the node corresponding to the configuration M moves into if the i-th symbol is a zero, and we connect the "1" outgoing edge to the node corresponding to the configuration M moves into if the i-th symbol is a one.

It is clear that on an input $(x, A(|x|))$, the just-described branching program for inputs of

length $n = |x|$ steps through exactly the sequence of configurations that M would on input x with advice $A(n)$, ending at *accept* if and only if M accepts x with advice $A(n)$. We can "hard-wire" the advice string $A(n)$ by short-circuiting any node labeled with a bit from $A(n)$ along the 0 or 1 edge, depending on the value of the bit in $A(n)$; this gives us our final polynomial-size branching program for inputs of length n.

In the other direction, it is clear that given a description of a polynomial-size branching program P as advice, we can simulate P on input x in log-space, as we need only keep track of our current location within the branching program and figure out the "0" or "1" neighbor based on reading a given bit from the read-only input tape. All of this requires only $O(\log |P|)$ bits of work space (we need to remember our current location in P; we need a counter of $\log n$ bits to scan the input for the i-th bit; and we need a counter for scanning the description of P to find the next node).

4. Suppose that SAT $\in$ **BPP**. We saw in class (using error reduction) that we can assume that there is a machine M with the following behavior:

$$\begin{aligned} x \in SAT &\Rightarrow \Pr_y[M(x,y)\text{accepts}] \geq 1 - 1/(4|x|) \\ x \notin SAT &\Rightarrow \Pr_y[M(x,y)\text{rejects}] \geq 1 - 1/(4|x|) \end{aligned}$$

We then run the **BPP** algorithm at most $2n$ times, filling in one bit of a satisfying assignment (if there is one) at each step. If at any step the algorithm gives inconsistent answers, we reject. Otherwise, if we get a purported satisfying assignment we accept if and only if it actually satisfies the formula. It is clear that we never incorrectly accept, as required. We simply need to show that we accept with probability at least $1/2$ when $x \in$SAT. In this case we accept if the algorithm never makes an error during the $\leq 2n$ calls to it. The probability that it makes an error on any one call is at most $1/4n$, so the probability that it makes an error on any one of the $2n$ calls is at most $1/(4n) \cdot 2n = 1/2$. Thus with probability at least $1/2$ it is correct on all calls, and we accept. Thus **NP** $\subseteq$ **RP**, and we already know **NP** $\supseteq$ **RP**, so we obtain **NP** = **RP** as required.

5. (a) Define $L = \{x : b(f^{-1}(x)) = 1\}$. This language is in **NP** because it is equal to $\{x : \exists y\ b(y) = 1 \text{ and } f(y) = x\}$. Notice that

$$\overline{L} = \{x : b(f^{-1}(x)) = 0\} = \{x : \exists y\ b(y) = 0 \text{ and } f(y) = x\},$$

so $L \in$ **coNP** as well. For both of these containments, we critically use the fact that f is a *permutation* – otherwise, e.g., we may have $x \notin L$ and yet for some $y \neq f^{-1}(x)$, $b(y) = 1$ and $f(y) = x$. Now, suppose that $L \in$ **BPP**. We know that **BPP** $\subseteq$ **P/poly**, so there is a polynomial size circuit that computes $b(f^{-1}(x))$ given x, which contradicts our initial assumptions on f and b, so it must be that $L \notin$ **BPP**.

(b) As suggested, define $g' = \{g'_n\}$ to be the function family obtained by truncating the output of each g_n to $t+1$ bits. In other words, there is some constant c for which each g'_n maps from $t = c \log_2 n$ to $t+1$ bits. Define the language

$$L = \bigcup_t \{x : \exists z\ \ g'_{2^t}(z) = x\}.$$

We first claim that $L \in \mathbf{E}$. Given x of length $t+1$, we only need to evaluate $g_{2^t}(z)$ at all z and check to see if the length $t+1$ prefix of any of these outputs equals x. Since g_{2^t} is computable in $\text{poly}(n) = 2^{O(t)}$ time, and we need to evaluate it 2^t times, the overall running time to decide L is $2^{O(t)}$, as claimed.

Now, we argue that L does not have small circuits. Suppose there is a circuit family $C = \{C_i\}$ deciding L. Notice that the number of strings of length i in L is at most 2^{i-1}, since there are only that many inputs of length $i-1$ to $g_{2^{i-1}}$. Thus

$$\Pr_{y \in \{0,1\}^i}[C_i(y) = 1] \leq 1/2.$$

On the other hand, since C decides L,

$$\Pr_{z \in \{0,1\}^{i-1}}[C_i(g'_{2^{i-1}}(z)) = 1] = 1.$$

We can also think of C_i as a circuit on more than i variables that only looks at its first i variables. Thus we conclude that

$$\left| \Pr_{y \in \{0,1\}^{2^{\delta(i-1)}}}[C_i(y) = 1] - \Pr_{z \in \{0,1\}^{c(i-1)}}[C_i(g_{2^{i-1}}(z)) = 1] \right| \geq 1/2,$$

which violates the second property of g, if the size of C_i is at most $2^{\delta(i-1)}$. We conclude that the circuit family $C = \{C_i\}$ does not have circuits of size $2^{\epsilon i}$, for, say, $\epsilon = \delta/2$ and sufficiently large i.