CS 151 Complexity Theory **Spring 2023**

Solution Set 5

Posted: May 18 *Chris Umans*

1. We are given a Boolean circuit C on n variables $x_1, x_2, \ldots, x_n$ with m $\wedge, \vee$ and $\neg$ gates. Our 3-CNF formula will have m auxiliary variables $z_1, z_2, \ldots, z_m$ in addition to the x variables, and we associate each z variable with one of the m gates. We want to enforce constraints so that any satisfying assignment to all of the variables will have the z variables taking on the value that the associated gates would output given the assignment to the x variables. We do this as follows:

   - for a $\neg$ gate associated with z_i, and with input w (which may be a z variable or an x variable), we enforce $\neg w \Leftrightarrow z_i$ by including the clauses $(w \vee z_i)$ and $(\neg w \vee \neg z_i)$.
   - for an $\wedge$ gate associated with z_i, and with inputs w and y (each of which may be a z variable or an x variable), we enforce $(w \wedge y) \Leftrightarrow z_i$ by including the clauses $(\neg w \vee \neg y \vee z_i)$, $(\neg z_i \vee w)$ and $(\neg z_i \vee y)$.
   - for an $\vee$ gate associated with z_i, and with inputs w and y (each of which may be a y variable or an x variable), we enforce $(w \vee y) \Leftrightarrow z_i$ by including the clauses $(\neg w \vee z_i)$, $(\neg y \vee z_i)$ and $(\neg z_i \vee w \vee y)$.

   Assume that z_m is the variable associated with the output gate. By construction our 3-CNF so far has the property that any assignment that satisfies the above clauses must assign to z_m the value that $C(x_1, x_2, \ldots, x_n)$ takes given the assignment to the x variables. We add a final clause (z_m). Now an assignment satisfies the formula if and only if the assignment sets the x variables in such a way that $C(x_1, x_2, \ldots x_n) = 1$. It is also easy to see that any assignment to the x variables for which $C(x_1, x_2, \ldots, x_n) = 1$ can be extended to an assignment to the x and z variables that satisfies all of the above clauses, by simply setting each z_i to the value the i-th gate is outputting in circuit C. Thus C is satisfiable if and only if the just-constructed 3-CNF formula is. If we call the 3-CNF formula ϕ, then we have, as desired:

   $$\exists z_1, z_2, \ldots, z_m \;\; \phi(x_1, x_2, \ldots, x_n, z_1, z_2, \ldots, z_m) = 1 \Leftrightarrow C(x_1, x_2, \ldots, x_n) = 1.$$

   For the second part, we first take C and add a $\neg$ gate to its output; call this circuit C'. Now applying the above transformation to C' gives a 3-CNF formula ϕ' with the property that:

   $$\exists z_1, z_2, \ldots, z_m \;\; \phi'(x_1, x_2, \ldots, x_n, z_1, z_2, \ldots, z_m) = 1 \Leftrightarrow C'(x_1, x_2, \ldots, x_n) = 1.$$

   Equivalently,

   $$\forall z_1, z_2, \ldots, z_m \;\; \phi'(x_1, x_2, \ldots, x_n, z_1, z_2, \ldots, z_m) = 0 \Leftrightarrow C'(x_1, x_2, \ldots, x_n) = 0.$$

   Let us define ϕ to be $\neg\phi'$, and note that (if we distribute the $\neg$) ϕ is a 3-DNF formula. We have:

   $$\forall z_1, z_2, \ldots, z_m \;\; \phi(x_1, x_2, \ldots, x_n, z_1, z_2, \ldots, z_m) = 1 \Leftrightarrow C'(x_1, x_2, \ldots, x_n) = 0.$$

Finally, by the definition of C' we have:

$$\forall z_1, z_2, \ldots, z_m \;\; \phi(x_1, x_2, \ldots, x_n, z_1, z_2, \ldots, z_m) = 1 \Leftrightarrow C(x_1, x_2, \ldots, x_n) = 1$$

as desired.

2. (a) Let M be the nondeterministic polynomial-time oracle TM deciding a language $L \in NP^A$. Given an input x, we will describe how to determine whether or not it is in L, in $BPP^{(\oplus P)^A}$. Define T_x to be the (possibly empty) set of accepting computation paths of M on input x (since M is fixed, each of these is described completely by a sequence of $m = \text{poly}(n)$ nondeterministic choices). Our BPP machine uses its randomness to produce, via the Lemma, a sequence of circuits $C_1, C_2, \ldots, C_k$ (each from an independent invocation of the procedure in the lemma with fresh random bits). Notice that determining whether

$$S_i = \{y : y \in T_x \text{ and } C_i(y) = 1\}$$

has odd size is a problem in $(\oplus P)^A$, because we can construct a nondeterministic oracle TM M_i that uses its nondeterminism to guess y and then enters state q_{accept} iff $y \in T_x$ and $C_i(y) = 1$ (and checking whether $y \in T_x$ can be done in polynomial time with access to oracle A). Our BPP machine thus makes k oracle calls and accepts iff $|S_i|$ is odd for some i.

If $x \notin L$, then regardless of the random bits of the BPP machine, $|S_i| = 0$ for all i, and so the BPP machine will always reject. If $x \in L$, then each S_i has at least a $1/(8m)$ chance of having size 1 (which is odd), and thus the probability we reject is at most $(1-1/(8m))^k \leq e^{-k/(8m)}$. Picking $k = 16m = \text{poly}(n)$ is sufficient to make this quantity less than $1/3$ as required by the definition of BPP.

(b) We prove the statement by induction on i. When $i = 1$ that implication is trivial. For general i, we have that the $NP^A \subseteq BPP^A$ implies that the "tower of height i" class is contained in NP^{BPP^A}, since by induction the oracle on the bottom "NP" (which is the "tower of height $i-1$") is contained in BPP^A.

We now argue that $NP^{BPP^A} \subseteq BPP^{NP^A}$. Given a language $L \in NP^{BPP^A}$ and an instance x of length n, let n^c be an upper bound on the running time of the NP^{BPP^A} machine. By error-reduction (just repetition is sufficient here), we can assume that the machine associated with the BPP^A oracle has error less than $1/(2^{n^c}3n^c)$. Our BPP oracle machine for deciding L does the following: it uses its randomness to choose a *single* string of length that exceeds the maximum randomness used by any of the invocations of the BPP^A oracle. Since there are at most n^c queries, each of length at most n^c, and the oracle uses only a polynomial in its input length number of random bits, this is poly(n). Now we will simulate the NP^{BPP^A} machine on input x *using this string as the randomness for every oracle query.* Because we made sure the error was less than $1/(2^{n^c}3n^c)$, we have, by a union bound (over the at-most 2^{n^c} computation paths of the NP machine and the at most n^c queries on each path), that the probability of *any* of these simulations being an error is less than $1/3$. After the randomness is fixed like this, each oracle query can be simulated in P^A and it is trivial that $NP^{P^A} \subseteq NP^A$. The overall simulation places L in BPP^{NP^A}, because as we noted, the error of the entire procedure is less than $1/3$.

So far, we have the 'tower of height i" class in BPP^{NP^A}. One further application of the hypothesis places it in BPP^{BPP^A}. Consider a language L in this class. We can use the same idea as in the previous paragraph: first reduce the error of the "base" BPP machine to less than (say) $1/6$, and then reduce the error of the machine associated with the oracle to less than $1/(6n^c)$ where n^c is an upper bound on the number of oracle queries made by the base BPP machine. Then even after taking a union bound over all the oracle queries, we get a probability of less than $1/6$ of error, which together with the error of $1/6$ of the base BPP machine, is less than the required $1/3$. The overall simulation place L in BPP^A, which yields the desired final result.

(c) Let L be a language in co-$\oplus P$, and let M be the associated nondeterministic TM, that makes m nondeterministic choices on each computation path. Define a new nondeterministic TM M' that makes $m+1$ nondeterministic choices on each computation path: the first nondeterministic choice determines whether to (a) simulate M or (b) simply make m nondeterministic 0/1 choices and enter q_{accept} iff they are all 0. Machine M' has exactly one more accepting computation path than M does (the one associated with choice (b) above when the m further nondeterministic choices are all 0). The existence of M' places L in $\oplus P$.

(d) Let L be a language in $(\oplus P)^{\oplus P}$, and let M be the nondeterministic oracle TM deciding L with access to an $\oplus P$ oracle. Let R be the TM associated with the $\oplus P$ language A that is the actual oracle attached to M, and let R' be the TM associated with the $\oplus P$ language co-A (using part (c)).

Define a nondeterministic TM M' as follows: on input x, M' uses its nondeterminism to guess a "transcript" of the operation of M on input x, on a single computation path. The transcript contains a sequence of nondeterministic choices made by M, together with a sequence of queries made to the oracle, and a sequence of yes/no answers. (Many of these transcripts will be inaccurate in the sense that they don't agree with the functioning of M on the specified computation path, with *the actual oracle* A answering queries, but *some* nondeterministic guesses produce "correct" transcripts). Let Y be the set of oracle queries that the transcript says are answered positively, and let N be the set of oracle queries that the transcript says are answered negatively. In "phase 2" M' guesses the nondeterministic choices of R on each of the queries in Y and the nondeterministic choices of R' on each of the queries in N. Machine M' enters q_{accept} if for every query in Y, these choices describe an accepting computation path in R, and for every query in N, these choices describe an accepting computation path in R', and if the transcript is *valid* (in the sense that M actually makes that sequence of queries on the specified computation path, when receiving the answers specified in the transcript) and it represents an accepting computation path of M (i.e., it leads to q_{accept}). Note that the decision to enter q_{accept} or not can be made in polynomial time given all the nondeterministic guesses made up to this point, and hence M' is a polynomial-time nondeterministic TM.

We claim that M' has an odd number of accepting computation paths on input x iff $x \in L$. The transcripts guessed by M' fall into two categories – those that are *valid* and lead to q_{accept} *and* whose "answers" contain the answers that oracle A *actually provides* on the specified queries, and all others. Each transcript in the first category gives rise to an odd number of accepting computation paths in "phase 2" of the operation of M',

because for each of the queries in Y, there are an odd number of accepting computations of R on that query, and for each of the queries in N, there are an odd number of accepting computation of R' on that query. These odd numbers multiply to give a total odd contribution from each transcript in the first category. Every other transcript either contributes 0 because it is not *valid*, or doesn't lead to q_{accept}, or it contributes an even number of accepting computations because at least one of the answers in the transcript disagrees with the A oracle answer, and so the associated query will have an even number of accepting computations of R (if it was in Y) or an even number of accepting computations of R' (if it was in N). And, all it takes is a single such query to multiply the total number of accepting computation paths associated with this transcript by an even number. All in all, the transcripts in the first category give rise to an odd number of accepting computation paths, and the transcripts in the second category give rise to an even number of accepting computation paths.

Finally, the transcripts in the first category are in one-to-one correspondence to the accepting computation paths of M on input x, so there are an odd number of them iff $x \in L$. Thus, the total number of accepting computation paths of M' on input x, if $x \in L$, is the sum of an odd number of odd numbers plus some even numbers. Considering this sum modulo 2, it is easy to see that this is an odd number. On the other hand, the number of accepting computation paths of M' on input x, if $x \notin L$, is the sum of an even number of odd numbers plus some even numbers, and this is clearly 0 modulo 2. We conclude that $L \in \oplus P$ as required.

(e) Recall that $PH = \cup_i \Sigma_i^P$, so it suffices to prove $\Sigma_i^P \subseteq BPP^{\oplus P}$ for all i. By part (a) with A being a language in $\oplus P$, we have that $NP^A \subseteq BPP^{(\oplus P)^A}$. By part (d) the oracle in $(\oplus P)^A$ can be replaced with a language B in $\oplus P$, yielding $NP^A \subseteq BPP^B$. Since both A and B are languages in $\oplus P$, which is a class that has complete languages, we can replace both with a single $\oplus P$-complete language C, yielding $NP^C \subseteq BPP^C$. Now, part (b) gives us that the "tower of height i" class, which trivially contains Σ_i^P, is contained in BPP^C, for each i. Language C is in $\oplus P$, so this is what was to be shown.

3. (a) For each $k = 1, 2, 3, \ldots, n-2$ we perform the following procedure:

- Pick a random $h \in \mathcal{H}_{n,k}$, and ask the **NP** oracle query: is there some $y \in \{0,1\}^k$ for which $|\{x : x \in S, h(x) = y\}| > n^4$?

Now, when k is such that $|S|/2^k > n^4$, we know by the pigeonhole principle that the answer to this query will be "yes." Now consider the *last* k for which this holds; i.e., the integer k for which

$$2^k < \frac{|S|}{n^4} \leq 2^{k+1}.$$

We claim that when we ask the query for $k+2$, with very high probability the answer will be "no." Since $2^{k+2} < 4|S|/n^4$, the Lemma applies, and tells us that for every $y \in \{0,1\}^{k+2}$,

$$\Pr_{h \in \mathcal{H}_{n,k+2}} \left[|\{x : x \in S \wedge h(x) = y\}| > 2 \cdot \frac{|S|}{2^{k+2}} \right] \leq 2^{-2n}.$$

Since $n^4 \geq |S|/2^{k+1} = 2|S|/2^{k+2}$, this implies:

$$\Pr_{h \in \mathcal{H}_{n,k+2}} \left[|\{x : x \in S \wedge h(x) = y\}| > n^4 \right] \leq 2^{-2n}.$$

By a union bound,

$$\Pr_{h \in \mathcal{H}_{n,k+2}} \left[\exists y \in \{0,1\}^{k+2} \text{ for which } |\{x : x \in S \wedge h(x) = y\}| > n^4 \right] \leq 2^n \cdot 2^{-2n} = 2^{-n}.$$

So, if we set k^* to be the first value for which we receive a "no" answer from the oracle, we know with very high probability that k^* will be either $k+1$ or $k+2$ (because by the time we get to $k+2$, we have shown that with high probability the answer will be "no"). Therefore we can satisfy the problem demands by outputting $k^* - 2$.

(b) Using the procedure from the first part, and k^* as defined above, we obtain a function $h : \{0,1\}^n \to \{0,1\}^{k^*}$ for which:

- $\forall y \in \{0,1\}^{k^*}$, the set $\{x : x \in S \wedge h(x) = y\}$ has size at most n^4 (because the **NP**-oracle query answer was "no" for k^*).
- with probability at least $7/8$, $2^{k^*-2} < |S|/n^4$, as required by (a).

We perform the following procedure: pick a random $y \in \{0,1\}^{k^*}$ and use the **NP**-oracle to enumerate the set $T_y = \{x : x \in S \wedge h(x) = y\}$. Choose a random number i between 1 and n^4 and output the i-th element of T_y, or "fail" if there is no i-th element. To connect this with the hint: we are thinking of each y as specifying a page of a notebook which has n^4 lines per page, and the elements T_y written on the first $|T_y|$ lines of page y.

The probability we output any given $x \in S$ is exactly $2^{-k^*} \cdot (1/n^4)$ – so conditioned on not failing, the output is exactly uniformly distributed on S. The probability that we output "fail" is exactly $1 - |S|/(2^{k^*}n^4)$. With probability at most $1/8$ the second item above fails to hold (and then we have no handle on the failure probability). Otherwise, we know that $|S| > 2^{k^*-2}n^4$, so the failure probability is at most

$$1 - \frac{|S|}{2^{k^*}n^4} < 1 - \frac{2^{k^*-2}n^4}{2^{k^*}n^4} = \frac{3}{4}.$$

Thus the overall failure probability is at most $3/4 + 1/8 = 7/8$, as required.