Technical Reports on Mathematical and Computing Sciences: TR-C168
title: BPP = PH by Polynomially Stringent Relativization
author: Jin-Yi Cai[1] and Osamu Watanabe[2]
affiliation:
1. Computer Sciences Dept., University of Wisconsin, Madison, WI 53706, USA (jyc@cs.wisc.edu)
2. Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology Meguro-ku Ookayama, Tokyo 152-8552, Japan (watanabe@is.titech.ac.jp)
acknowledgments to financial supports:
1. Supported in part by NSF grants CCR-0208013 and CCR-0196197 and U.S.-Japan Collaborative Research NSF SBE-INT 9726724.
2. Supported in part by the JSPS/NSF Collaborative Research 1999 and by the Ministry for Education, Grant-in-Aid for Scientific Research (C), 2001.

Abstract. For relativized arguments, we propose to restrict oracle queries to "stringent" ones; for comparing the power of two machine models relative to some oracle set, stringent relativization is to restrict machines of both types to ask queries on the same segment of the oracle. In particular, for investigating polynomial-time (or polynomial-size) computability, we propose *polynomial stringency*, bounding query length to any fixed polynomial of input length. Under such stringent oracle access, we show an oracle G such that $\mathrm{BPP}^G = \mathrm{PH}^G$.

1 Introduction

In [CW02], the authors proposed "stringent relativization" for investigating a fine relationship between complexity classes such as $\mathrm{NTIME}(n^{k_1})$ and $\mathrm{SIZE}(n^{k_2})$ (= the class of problems solvable by $\mathcal{O}(n^{k_2})$ size circuits). But considering "stringent relativization" is meaningful even for more general relationships such as P vs. NP. In this paper, we propose "polynomially stringent relativization" for discussing relations between various complexity classes. As a technical result, we prove that BPP = PH by polynomially stringent relativization, and as a corollary, we show that PH $\subseteq$ P/poly by polynomially stringent relativization, giving a simpler proof of the main result in [CW02]. We note that it is open how to exhibit a stringent relativization with which P = PSPACE.

We first review some standard relativized argument and explain our motivation of proposing stringent relativization. Below we use standard complexity classes and standard results in computational complexity theory without explaining them. See, e.g., text books [BDG89, DK00] for the explanation.

Consider the standard *collapsing* relativized argument for the P vs. NP conjecture; that is, showing an oracle A satisfying $\mathrm{P}^A = \mathrm{NP}^A$. For this set A, we may consider any complete set for PSPACE. For example, let us consider the problem QBF. From its PSPACE-completeness, we have PSPACE $\subseteq \mathrm{P}^{\mathrm{QBF}}$. On the other hand, $\mathrm{NP}^{\mathrm{QBF}} \subseteq$ PSPACE. Thus, we have $\mathrm{NP}^{\mathrm{QBF}} \subseteq \mathrm{P}^{\mathrm{QBF}}$; hence, $\mathrm{P}^{\mathrm{QBF}} = \mathrm{NP}^{\mathrm{QBF}}$.

While this is a valid argument, in order to see how queries are used for this collapsing argument, let us consider a bit more in detail the simulation of any NP query machine $\mathcal{Q}_0$ by

some P query machine $\mathcal{Q}_1$ relative to QBF. Assume that $\mathcal{Q}_0$ is $\mathcal{O}(n^k)$-time bounded machine. Then one can easily design some PSPACE machine $\mathcal{M}_0$ simulating $\mathcal{Q}_1^{\mathrm{QBF}}$; more precisely, for any input x of length n, $\mathcal{M}_0(x)$ simulates $\mathcal{Q}_0^{\mathrm{QBF}}(x)$ using $\mathcal{O}(n^k)$ space. Then by using the standard reduction, we can express the computation $\mathcal{M}_0(x)$ in terms of some QBF formula F_x. That is, $\mathcal{M}_0$ accepts x if and only if $F_x \in$ QBF. Thus, by asking one query (namely, F_x) to the oracle QBF, one can simulate $\mathcal{M}_0(x)$ (which is $\mathcal{Q}_0^{\mathrm{QBF}}(x)$). Since F_x is polynomial-time computable from x, this proves that $\mathcal{Q}_0^{\mathrm{QBF}}$ is simulated by some deterministic polynomial-time machine $\mathcal{Q}_1$ asking one query to QBF.

Now compare the length of queries asked by these two machines $\mathcal{Q}_0$ and $\mathcal{Q}_1$. For any input x of length n, let the queries that $\mathcal{Q}_0$ makes on x be up to ℓ in length, which is bounded by $O(n^k)$. On the other hand, $\mathcal{Q}_1$ asks F_x, which is a QBF formula of length $> \ell$. Thus, so long as a standard reduction[1] to QBF is used to define F_x, the simulating machine $\mathcal{Q}_1$ asks strictly longer queries than those asked by the simulated machine $\mathcal{Q}_0$. We find that most relativized *collapsing* arguments follow more or less the same style; that is, a simulating machine asks longer queries than those asked by a simulated machine, and a collapsing result is proved by using (sometimes implicitly) this superiority of the simulating machine to the simulated one. We think that this type of simulation overstates our intuition on relativization; or at least, this notion can be made more strict to better reflect our intuition.

Intuitively, for each oracle set X, the relative computation model allowing oracle queries to X provides a "relativized complexity world" where all computation is the same as our real world except that one can use some special set of instructions, i.e., queries to the oracle set X. From this point of view, the argument that we have just seen above is not a natural comparison because the simulating machine can access some different part of the oracle. To express this intuition of relativization, we propose a new model of stringent relativization in the paper. We propose to restrict relativized computation so that both machines (i.e., simulated and simulating machines) can see the same segment of an oracle set. More technically, our restriction is to limit query length to any *fixed* polynomial for both query machines, which we call "polynomially stringent" relativization.

We define this notion precisely. First consider the following restricted relativized computation. (Below by a "length bound", we mean a function from $\mathbf{N}$ to $\mathbf{N}$ that is "reasonable" such as time/space constructible in each context.)

Definition 1. Let Q be any query machine (or circuit family with query gates) of any type. For any length bound $\ell(n)$, we say Q is executed *under* $\ell(n)$ *length restricted relativization* (relative to an oracle set X) if any query of length $> \ell(n)$ gets a 'no' answer (while any query of length $\leq \ell(n)$ gets an answer consistent with X) on the computation of Q of any input of length n.

Roughly speaking, for stringent relativizations, we propose to compare two machines under length restricted relativization. That is, compare query machines Q_1 and Q_2 executed under length restricted relativization, and discuss, for example, whether Q_1 simulates Q_2 relative to some oracle X under length restricted relativization.

[1]It seems that any reduction to QBF cannot be so efficient. In fact, as far as the authors know, there seems no PSPACE complete set X that allows us to simulate any NP query machine $\mathcal{Q}_0$ executed relative to X by some P query machine $\mathcal{Q}_1$ asking the same length of queries to X.

While two machines can be compared in this way, some care is needed in order to obtain a reasonable definition to compare two complexity classes. Here we propose the following definition for a basic "stringent relativization" notion.

Definition 2. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be any complexity classes, and let $\ell(n)$ be any length bound.
(1) For any oracle set G, we say that containment $\mathcal{C}_1 \subseteq \mathcal{C}_2$ is shown *by $\ell(n)$ length stringent relativization relative to G* (or, in short, $\mathcal{C}_1^G \subseteq \mathcal{C}_2^G$ *under $\ell(n)$ length stringent oracle access*) if for any query machine $\mathcal{Q}_1$ for $\mathcal{C}_1$, there exists some query machine $\mathcal{Q}_2$ for $\mathcal{C}_2$ such that $L(\mathcal{Q}_2^G) = L(\mathcal{Q}_1^G)$ holds, when both machines are executed under $\ell(n)$ length restricted relativization.
(2) For any oracle set G, we say that separation $\mathcal{C}_1 \not\subseteq \mathcal{C}_2$ is shown *by $\ell(n)$ length stringent relativization relative to G* (or, in short, $\mathcal{C}_1^G \not\subseteq \mathcal{C}_2^G$ *under $\ell(n)$ length stringent oracle access*) if for some query machine $\mathcal{Q}_1$ for $\mathcal{C}_1$, no query machine for $\mathcal{C}_2$ exists such that $L(\mathcal{Q}_2^G) = L(\mathcal{Q}_1^G)$ holds, when both machines are executed under $\ell(n)$ length restricted relativization.
Remark. We will use symbol G to denote an oracle for stringent relativization.

Clearly, bounding query length is vacuously true, in the above definition, if the length bound $\ell(n)$ is larger than the bound allowed for one of the complexity classes $\mathcal{C}_1$ and $\mathcal{C}_2$. Thus, one should choose an appropriate query bound. In particular, for relationships between polynomially resource bounded classes, we propose the following stringency.

Definition 3. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be any complexity classes. We say that their relationship (i.e., containment or separation) is shown *by polynomially stringent relativization* if for any length bound $\ell(n)$, $\omega(n) \leq \ell(n) \leq n^{O(1)}$, there exists some oracle relative to which the relationship is shown by $\ell(n)$ length stringent relativization.
Remark. An oracle can be (in most cases "should be") defined for each length bound $\ell(n)$. But since each oracle for $\ell(n)$ is defined similarly (in most cases), we will simply state it as if a single oracle as in, e.g., "$\mathcal{C}_1^G \subseteq \mathcal{C}_2^G$ by polynomially stringent relativization."

Notice that, in our definition, we compare complexity classes for each fixed query length bound $\ell(n)$. The stringency would become much weaker if we can use different length bounds for two machines. For example, consider the relationship NP $\subseteq$ P. If we can use *some* polynomial as a query length bound for the deterministic machine simulating a given $p(n)$-time bounded nondeterministic machine, then the standard relativization that explained above works by using $q(n)$ $(> p(n))$ as a length bound.

2. A Simple Example: $\mathrm{NP}^G \subseteq \mathrm{P}^G/\mathrm{poly}$

For an example of polynomially stringent relativization, we show the following result. (The argument shown below is well-known in computational complexity, but we state the proof to illustrate the difference from the usual "nonstringent" relativizations in a simple setting.)

Theorem 1. NP $\subseteq$ P/poly by polynomially stringent relativization. More precisely, for any polynomial length bound $\ell(n) \geq (2+\epsilon)n$ for some ϵ, some oracle G can be constructed so that $\mathrm{NP}^G \subseteq (\mathrm{P}/\mathrm{poly})^G$ holds by $\ell(n)$ length stringent relativization.

Notice first that for the polynomially stringent relativization, a higher collapse is not immediate from a lower one. For example, $\mathrm{NP}^G \subseteq \mathrm{P}^G/\mathrm{poly}$ does not necessarily imply by stringent relativization $\mathrm{P}^{\mathrm{NP}^G} \subseteq \mathrm{P}^G/\mathrm{poly}$, ..., $\mathrm{PH}^G \subseteq \mathrm{P}^G/\mathrm{poly}$, etc.

Proof. For simplicity, we construct an oracle G for a length bound $\ell(n) = 3n$.

Consider a standard enumeration $\mathcal{Q}_1, \mathcal{Q}_2, \ldots$ of all NP query machines. We define G by stages. For each n, we define $G^{=3n} \cup G^{=3n+1} \cup G^{=3n+2}$ so that one can easily compute $\mathcal{Q}_i^G(x)$, for any machine $\mathcal{Q}_i$ in the first k machines in the enumeration and for all input string x of length n. (In fact, in our construction, $G^{=3n+1} \cup G^{=3n+2}$ is the empty set $\emptyset$. That is, only strings of length $3n$ are added in G.)

Let us first fix some $\mathcal{Q}_i$ and discuss how the oracle set G is defined. Let p_i be the polynomial time bound for $\mathcal{Q}_i$. Here we may assume that n is sufficiently large so that $p_i(n)2^n < 2^{2n}$ holds.

Starting from 0^n, we examine lexicographically each input $x \in \{0,1\}^n$ and determine the membership of some length $3n$ strings into G. (We may assume that $G^{<3n}$ has been fixed before this stage.) Suppose that we have examined strings before x; at this point, $G^{=3n}$ is only partially fixed. Consider all extensions in $G^{=3n}$ consistent with all the fixed strings in or out of $G^{=3n}$ so far, we look for some nondeterministic path that accepts x if G is extended appropriately. If such an accepting path exists, then extend G consistent with this accepting path (i.e., fix the membership of some strings in $\{0,1\}^{=3n}$ in $G^{=3n}$), and proceed to the next input string. (Note that this incurs only at most $p_i(n)$ many additional fixed positions.) Otherwise, i.e., if no consistent extension has such an accepting path, then Q_i rejects x as long as G is extended consistent with all the fixed positions so far. In this case, we simply proceed to the next input string.

Suppose that $G^{=3n}$ has been partially defined as above by examining all input strings of length n. Since the machine asks at most $p_i(n)$ strings on each nondeterministic path, for each input string, we only have to fix the membership of at most $p_i(n)$ strings of length $3n$ to force $\mathcal{Q}_i$ to accept x (if it is indeed possible). Hence, at most $p_i(n)2^n$ strings of length $3n$ are fixed after examining all 2^n input strings of length n. Therefore, there is some string u_i of length $2n$ such that no string in $u_i\{0,1\}^n$ is fixed when defining G. We simply use this part to encode the information whether $\mathcal{Q}_i^G$ accepts x or not. That is, for each $x \in \{0,1\}^n$, a string $u_i x$ is put into G if $\mathcal{Q}_i^G$ accepts x; no other string is added to define $G^{\leq 3n}$. Notice that adding these unfixed strings to G does not change the acceptance of $\mathcal{Q}_i^G$ on any input of length n. On the other hand, since $u_i x$ is in G if and only if $\mathcal{Q}_i^G$ accepts x, one can easily compute $\mathcal{Q}_i^G(x)$ by checking whether $u_i x$ is in G. For this computation, it is necessary to know this key u_i, but this information can be hardwired in a polynomial-size circuit.

Precisely speaking, we need to simulate all k machines $\mathcal{Q}_1$, $\mathcal{Q}_2$, ..., $\mathcal{Q}_k$ for some k. By an argument similar to the above, such simulation is possible if n is large enough so that $\sum_{1 \leq i \leq k}(p_i(n) + 1)2^n < 2^{2n}$. we can encode all results by using k segments, $u_1\{0,1\}^n$, ..., $u_k\{0,1\}^n$, with strings u_1, ..., u_k in $\{0,1\}^{2n}$. Then a simple circuit with the key string u_i hardwired can compute $\mathcal{Q}_i^G(x)$ for all $x \in \{0,1\}^n$. □

The main utility of relativization is to show the inadequacy of certain proof techniques. Certainly the more "stringent" a requirement we place on the type of relativization, the stronger the result will be, and perhaps it says more about the infeasibility of certain techniques. Imagine there is a claim of $\mathrm{NP} \not\subseteq \mathrm{P}/\mathrm{poly}$; we may consider the following three proof types.

1. A proof totally specific to a certain NP set and/or some properties of polynomial-size circuits.
2. A proof that can be carried through even if we allowed NP machines and circuits to access any but the same segment of any oracle.
3. A proof that is more general and that can go through even if we allowed NP machines and circuits to access different segments of the oracle.

Any relativized NP $\subseteq$ P/poly result says nothing about the first possibility. A standard relativized result can only rule out the third possibility. On the other hand, our above result rules out both the second and the third possibilities.

3. The Main Result: $\mathrm{BPP}^G = \mathrm{PH}^G$

We now show $\mathrm{BPP}^G = \mathrm{PH}^G$ by polynomially stringent relativization. Technically, we prove Theorem 2 below. The construction of a polynomially stringent oracle for $\mathrm{BPP}^G = \mathrm{PH}^G$ can be done by a standard argument following Theorem 2, since PH is a countable union of Σ_d^{p}.

Theorem 2. For any $d \geq 1$, $\Sigma_d^{\mathrm{p,G}} \subseteq \mathrm{BPP}^G$ by polynomially stringent relativization. More specifically, for any polynomial length bound $\ell(n) \geq 3n$, we can construct an oracle G for which the following holds by $\ell(n)$ length stringent relativization: Any Σ_d^{p} machine $\mathcal{Q}_1$ with $\mathcal{O}(n^k)$ running time can be simulated (relative to G) by some BPP machine whose running time is bounded by $\mathcal{O}(n^{2dk})$.

By the standard argument (see, e.g., p. 142 of [BDG89]) for converting a given BPP computation to a circuit, the following corollary is immediate.

Corollary 3. $\mathrm{PH}^G \subseteq \mathrm{P}^G/\mathrm{poly}$ by polynomially stringent relativization. More specifically, for any $d \geq 1$ and any polynomial length bound $\ell(n) \geq 3n$, we can construct an oracle G for which the following holds by $\ell(n)$ length stringent relativization: Any Σ_d^{p} machine $\mathcal{Q}_1$ with $\mathcal{O}(n^k)$ running time can be simulated (relative to G) by some circuit of size $\mathcal{O}(n^{2dk})$.

Before giving a precise argument, we explain the outline of our proof. In the following, let us fix the query length bound to $\ell(n) = 3n$. It is easy to see the following argument works for any larger bound.

Fix any Σ_d^{p} query machine $\mathcal{Q}$ and any sufficiently large input length n. We describe how to construct an oracle G with which some polynomial-time probabilistic machine can simulate the computation of $\mathcal{Q}$ on inputs of length n by asking queries of length $\ell(n) = 3n$. Precisely speaking, we need to construct an oracle that enables the simulation of the first, say, i machines $\mathcal{Q}_1, \ldots, \mathcal{Q}_i$ in the enumeration of all Σ_d^{p} machines; but such a generalization is easy and we leave it to the reader. Thus, from now on, we are concerned with the simulation of a single $\mathcal{Q}$ on 2^n inputs of length n.

Let m be an integer bounding $\mathcal{Q}$'s running time on inputs of length n, where $m = O(n^k)$ for some $k \geq 1$. Let $\ell = \ell(n)$ $(= 3n)$ and $L = 2^\ell$. Assume that G has been already defined up to length $\ell - 1$ in the earlier stages of our oracle construction; then our task is to assign strings in $\{0,1\}^\ell$ to G. Enumerate the strings in $\{0,1\}^\ell$ and introduce Boolean variables $z_1, \ldots, z_L$, where each z_i indicates whether the ith string in $\{0,1\}^\ell$ is put into G. In the following, we

will explain our construction mainly using these Boolean variables. Let Z denote the set of all Boolean variables $z_1, \ldots, z_L$.

For any input string $x \in \{0,1\}^n$, consider the computation of $\mathcal{Q}(x)$ with an oracle G that is defined only up to length $\ell - 1$. Then the computation $\mathcal{Q}^G(x)$ is regarded as a function from $z_1, \ldots, z_L$ to $\{0,1\}$. Furthermore, since $\mathcal{Q}$ is a Σ_d^{p} machine, by a standard interpretation of the Σ_d^{p} query computation, we may regard $\mathcal{Q}^G(x)$ as a depth $d+1$ circuit C_x on input variables $z_1, \ldots, z_L$, of size at most $m2^m$ and bottom fan-in at most m. Our first step is to assign a random restriction ρ to $z_1, \ldots, z_L$ of an appropriate probability $p_0 = \Pr[\, z_i = * \,]$. Then by a decision tree version of the Switching Lemma (see Lemma 1), we can prove that with high probability, the circuit is reduced to a small depth (i.e., depth $t = 2m$) decision tree. In fact, by choosing $p_0 = 1/(20m)^d$, we can show that with high probability, a random restriction converts circuit C_x to a depth t decision tree T_x for *all* 2^n input strings x.

Fix one such random restriction ρ_0 that also satisfies certain other "balancing" conditions to be explained shortly. Assume in the decision trees, every left branch is labeled 0 for some Boolean variable, then the value of a depth t decision tree is determined by assigning 0 to all variables appearing in the path from its root to its leftmost leaf and that the number of such variables is at most t. Thus, after the random restriction, we can fix the value of all decision trees T_x's (hence, all circuits C_x's) by assigning 0 to at most $t2^n$ $(= 20m2^n)$ variables. (In the following, these variables are called *decision tree variables*.) Since there are L $(= 2^{3n})$ variables in Z, we still have many unassigned variables. What is crucial here is that no matter how we assign values to these leftover variables, the values of all circuits C_x's (in other words, the outputs of $\mathcal{Q}$ on all x's) will not be altered at all. Our idea is to use these leftover variables to encode all circuit values.

For each $x \in \{0,1\}^n$, we use a subset $x\{0,1\}^{2n}$ of $\{0,1\}^\ell$ to encode the (already fixed) value of C_x. (Recall that $\ell = 3n$.) Here for each $u \in \{0,1\}^{2n}$, we denote by z_{xu} the Boolean variable indicating the membership of xu to the oracle G, and let Z_x be the set of all such Boolean variables for the set $x\{0,1\}^{2n}$. Our plan is to assign the value of C_x to every one of these variables in Z_x. That is, assign $z_{xu} = 0$ (resp., $z_{xu} = 1$) for all $u \in \{0,1\}^{2n}$ if C_x yields 0 (resp., 1), *unless* the variable is already assigned 0 or 1 by ρ_0, or it is a decision tree variable, in which case it has been assigned 0.

Now we explain the "balancing" requirement on ρ_0. We choose our random restriction ρ_0 so that its assignment is not biased and quite close to the average. Recall that a random restriction assigns 0 or 1 to each variable with the same probability, and that it leaves each variable unassigned with probability p_0 $(= 1/(20m)^d)$. We choose ρ_0 so that, for every $x \in \{0,1\}^n$, (i) it leaves about $p_0 \cdot |Z_x|$ $(= 2^{2n}/(20m)^d)$ variables unassigned, and (ii) it assigns 0 and 1 to almost equally to variables in Z_x. By a standard probabilistic argument, we can prove that such a random restriction exists. Recall that there are at most $t2^n$ $(= 20m2^n)$ decision tree variables, which is exponentially smaller than $2^{2n}/(20m)^d$; hence, even if all decision tree variables appeared in one Z_x their proportion would be smaller than, say, $1/(20(20m)^d)$. Thus, for each x, by assigning the value of C_x, say, 1, to all leftover variables in Z_x, we can expect that the proportion of variables in Z_x assigned 1 is about $1/(5(20m)^d)$ $(= 1/\mathrm{poly}(n))$ advantage over those variables assigned 0. This is how we define $G^{=\ell}$. With this oracle G, one can probabilistically determine whether $\mathcal{Q}^G$ accepts x for each x as follows; simply ask polynomially many *randomly generated* queries of the form xu to G, and accepts x (resp., rejects x) if the

majority of them gets 'yes' (resp., 'no') answer.

Precise Argument

We first state the decision tree version of the Switching Lemma.

The decision tree complexity of a Boolean function f, denoted by $\mathrm{DC}(f)$, is the smallest depth of a Boolean decision tree computing the function. It can be shown easily that if $\mathrm{DC}(f) \leq t$, then f can be expressed both as an AND of OR's as well as an OR of AND's, with bottom fan-in at most t. Moreover, what is crucial for our argument is the following property: If $\mathrm{DC}(f) \leq t$, then the value of f is determined, when one assigns values to at most t variables.

Adapting Håstad's proof [Hås86a] to the decision tree model, one can prove the following. (The proof will be given in detail in a journal version of [CW02], and it is omitted here.)

Lemma 1. For any depth $d+1$ Boolean circuit C on $z_1, \ldots, z_L$, with bottom fan-in at most t,

$$\Pr[\ \mathrm{DC}(C\,|_\rho) \geq t\ ]\ \leq\ \frac{\mathrm{size}(C)}{2^t},$$

where ρ is a random restriction with the parameter $p = \Pr[z_i = *] = 1/(10t)^d$.

Now we specify the parameters and the conditions explained above, and describe our proof precisely. Here as in the above we focus on the simulation of some Σ_d^{p} query machine $\mathcal{Q}$ on 2^n inputs of length n for sufficiently large n.

For our random restriction ρ_0, we choose $p_0 = 1/(20m)^d$ and require that it satisfies the following three conditions for every $x \in \{0,1\}^n$. (Let N denote $|Z_x|$ $(= 2^{2n})$.)

(a) After the assignment by ρ_0, the circuit C_x is reduced to a decision tree of depth $\leq t = 2m$.
(b) $|\{\, z_{xu} \in Z_x \,:\, \rho_0(z_{xu}) = * \,\}| \geq p_0 N/2$.
(c) The absolute value of the difference of the cardinalities $|\{\, z_{xu} \in Z_x \,:\, \rho_0(z_{xu}) = 1 \,\}|$ and $|\{\, z_{xu} \in Z_x \,:\, \rho_0(z_{xu}) = 0 \,\}|$ is at most $p_0 N/4$.

By using Lemma 1 and the Chernoff bound, we can prove the Claim below, which guarantees that some random restriction ρ_0 satisfies the requirement. Then we can define G as argued above. That is, for any $w \in \{0,1\}^\ell$ $(= xu$ for some $x \in \{0,1\}^n$ and $u \in \{0,1\}^{2n})$, first if $\rho_0(z_w) \neq *$, then put w into G if and only if $\rho_0(z_w) = 1$; if $\rho_0(z_w) = *$ and w is a decision tree variable, then do not put w into G; otherwise, put w into G if and only if the value of (already determined) C_x is 1. On the other hand, to determine whether a given $x \in \{0,1\}^n$ is accepted by $\mathcal{Q}^G$, simply generate, say, $100/p_0^2$ queries of the form xu and outputs the majority of the answers from the oracle G on these queries.

Let us check whether this construction and our usage of G work. Consider any $x \in \{0,1\}^n$ such that $\mathcal{Q}^G$ accepts x, and estimate the following difference Δ. (The other case, i.e., the case when $\mathcal{Q}^G$ rejects x, can be checked similarly.)

$$\Delta\ =\ |x\{0,1\}^{2n} \cap G^{=3n}| - |x\{0,1\}^{2n} \cap (\{0,1\}^{3n} - G^{=3n})|.$$

From our requirement above on ρ_0 (i.e., the condition (a)), there are at least $p_0 N/2$ (where $N = 2^{2n}$) elements of $x\{0,1\}^{2n}$ left undetermined by ρ_0. If all of them were put into G, then Δ would be at least $p_0 N/2 - p_0 N/4 = p_0 N/4$ (by the condition (c)). In the real case, some of

them are not put into G in order to determine the decision tree values; but since there at most $t2^n$ $(= 2m2^n < p_0N/20)$ such elements in total, we still have $\Delta \geq p_0N/4 - p_0N/20 = p_0N/5$. Thus, for a randomly chosen $xu \in x\{0,1\}^{2n}$, the probability of $xu \in G$ is at least $p_0/5$ higher than that of $xu \notin G$. Thus, by using the Chernoff bound (stated below) again, we can bound the error probability of our simple sampling procedure by $4e^{-4}$. This completes our proof.

Now it suffices to prove the following claim.

Claim. Under our choice of parameters, the probability that a random restriction ρ satisfies (a) $\sim$ (c) is not zero.

Proof. For each one of the conditions (a) $\sim$ (c) and for every $x \in \{0,1\}^n$, we show that the probability that a random restriction fails to satisfy it is less than $2^{-(n+2)}$. Then the probability that *some* x fails to satisfy *some* of these conditions is less than $3 \cdot 2^n \cdot 2^{-(n+2)} < 1$.

First for some $x \in \{0,1\}^n$, consider the condition (a). Recall that C_x is a circuit of size $\leq m2^m$, depth $\leq d+1$, and bottom fan-in at most m. Then from our choice of t and p_0 and Lemma 1, it follows that the probability that $\mathrm{DC}(C_x\,|_\rho) \geq t$ is less than 2^{-m}, which is smaller than $2^{-(n+2)}$. (We may assume that m, $\mathcal{Q}$'s running time on inputs of length n, is larger than $n+2$.)

For the conditions (b) and (c), let us use the following version of the Chernoff bound (see, e.g., Appendix A of [AS00]): Let $X_1, \ldots, X_N$ be independent random variables taking 0 or 1, where $\Pr[X_i = 1] = p$. Let $X = \sum_{i=1}^N X_i$ and $\mu = \mathrm{Exp}[X]$ $(= pN)$. Then for any $a > 0$, we have

$$\Pr[\ |X - \mu| > a\ ] \ < \ 2e^{-2a^2/N}.$$

Now consider the condition (b) for some $x \in \{0,1\}^n$. Let X_i be a random variable indicating the event that a random restriction assigns $*$ to the ith variable in Z_x, where $1 \leq i \leq N$ $(= 2^{2n})$. Let $X = \sum_{i=1}^N X_i$. Then we have $\Pr[X_i = 1] = p_0$ $(= 1/\mathrm{poly}(n))$, and $\mu_0 = \mathrm{Exp}[X] = p_0N$. Hence, the probability that a random restriction fails to satisfy the condition (b) is estimated as follows for sufficiently large n.

$$\Pr[\ X < \frac{p_0N}{2}\ ] \ \leq \ \Pr[\ |X - \mu_0| > \frac{\mu_0}{2}\ ] \ < \ 2e^{-p_0^2N/2} \ < \ 2^{-(n+2)}.$$

Consider the condition (c) for some $x \in \{0,1\}^n$. Let X_i^+ (resp., X_i^-) be a random variable indicating the events that a random restriction assigns 1 (resp., 0) to the ith variable in Z_x. Let $X^+ = \sum_{i=1}^N X_i^+$ and $X^- = \sum_{i=1}^N X_i^-$, where $N = 2^{2n}$. Then we have $\Pr[X_i^+ = 1] = \Pr[X_i^- = 0]$ $= (1-p_0)/2$ $(> 1/4)$, which we denote by p_1, and $\mathrm{Exp}[X^+] = \mathrm{Exp}[X^-] = p_1N$, which we denote by μ_1. Then for sufficiently large n, the probability that a random restriction fails to satisfy the condition (c) is bounded by

$$\Pr[\ |X^+ - \mu_1| > \frac{p_0N}{8} \ \vee \ |X^- - \mu_1| > \frac{p_0N}{8}\ ] \ < \ 4e^{-p_0^2N/32} \ < \ 2^{-(n+2)}.$$

□

References

[AS00] N. Alon and J. Spencer, *The Probabilistic Method*, John Wiley & Sons, Inc., 2000.

[BDG89] J. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity I & II*, Springer, 1989 and 1990.

[Cai86] J-Y. Cai, With probability one, a random oracle separates PSPACE from the polynomial-time hierarchy, in *Proc. 18th ACM Sympos. on Theory of Comput.*, 21–29, 1986. (The final version appeared in *J. Comp. Syst. Sci.*, 38(1), 68–85, 1989.)

[CW02] J-Y. Cai and O. Watanabe, On proving circuit lower bounds against the polynomial-time hierarchy, subimitted.

[DK00] D. Du and K. Ko, *Theory of Computational Complexity*, John Wiley & Sons, Inc., 2000.

[Hås86a] J. Håstad, Almost optimal lower bounds for small depth circuits, in *Proc. 18th ACM Symposium on Theory of Computing* (STOC'86), ACM, 6–20, 1986.