# Relativized Collapsing Results under Stringent Oracle Access

LA-1

Jin-Yi Cai* Osamu Watanabe†

## 1 Introduction

The P vs. NP conjecture is one of the most important conjectures in mathematical science. Though we are proud of this conjecture originated in Theoretical Computer Science, it is pity that we have not been able to find any clue solving this conjecture. In fact, it has been widely believed that this and similar related conjectures cannot be proved by any of our known proof techniques.

This pessimistic belief is supported by *relativized results*. That is, we can relativize the conjecture both ways. More specifically, there exist two oracles $A$ and $B$ such that $\mathrm{P}^A = \mathrm{NP}^A$ (the *collapsing*) holds and $\mathrm{P}^B \neq \mathrm{NP}^B$ (the *separation*) holds. 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 intuition, it is said that almost all known proof techniques are *relativizable*; that is, they are applicable in such relativized worlds. Therefore, having the above oracles $A$ and $B$ means that almost all known proof techniques are hopeless for resolving the conjecture.

In this paper, however, we point out that some of those relativized results (the collapsing ones) are not stringent enough compared with our intuition explained above. We introduce more stringent oracle access and propose to prove such relativized results under this type of oracle access. We then prove, for example, that $\mathrm{BPP}^C = \mathrm{NP}^C$ with some oarcle $C$ under the stringent oracle access.

To explain the problem of existing relativized arguments, we review the relativized collapsing result for the P vs. NP conjecture. For an oracle $A$ satisfying $\mathrm{P}^A = \mathrm{NP}^A$, we may consider any complete set for PSPACE. For example, let us consider the following canonical complete set for $A$.

$$A = \{(\mathcal{M}, x, 0^s) : M \text{ accepts } x \text{ by using } s \text{ space}\}.$$

Where $\mathcal{M}$, $x$, and $0^s$ are respectively the description of a deterministic Turing machine, a string in $\{0,1\}^*$, and a sequence of $s$ 0's. We assume that $(\mathcal{M}, x, 0^s)$ is encoded as a string in $\{0,1\}^*$ in some reasonable way.

It is easy to check that $\mathrm{P}^A = \mathrm{NP}^A$. That is, relative to the oracle $A$, every oracle NP-machine can be simulated by some oracle P-machine. To see how queries are used for this collapsing argument, we consider the simulation of any oracle NP-machine $\mathcal{Q}_0$. Note that the computation of $\mathcal{Q}_0^A$ can be simulated by some PSPACE-machine $\mathcal{M}_0$. Let $s_0$ be a polynomial space bound for $\mathcal{M}_0$. Then the construction of an oracle P-machine $\mathcal{Q}_1$ simulating $\mathcal{Q}_0$ is easy. On a given input $x$ of length $n$, $\mathcal{Q}_1$ simply asks the query $(\mathcal{M}_0, x, 0^{s_0(n)})$ to $A$, and then outputs its answer. It is easy to see that $\mathcal{Q}_1^A$ simulates $\mathcal{Q}_0^A$ correctly.

Notice here that the query made by $\mathcal{Q}_1$ is much longer than queries asked in the simulated $\mathcal{Q}_0^A$-computation. For example, assume that, for each input $x$ of length $n$, $\mathcal{Q}_0^A(x)$ always asks a query of length $\ell_0(n)$ to $A$. Then the PSPACE-machine $\mathcal{M}_0$ simulating $\mathcal{Q}_0^A$ needs space more than $\mathcal{O}(\ell_0(n))$. That is, $s_0(n)$ cannot be $\mathcal{O}(\ell_0(n))$. Then the length of the query $(\mathcal{M}_0, x, 0^{s_0(n)})$ made by the oracle P-machine $\mathcal{Q}_1$ is much larger than $\ell_0(n)$. That is, the oracle P-machine is allowed to ask much longer queries than those asked by the simulated machine $\mathcal{Q}_0$. Thus, this type of simulation is slightly different from our intuitive understanding of relativized arguments.

In this paper, we propose a relativized argument that formulate our intuition more naturally, which is formally stated as follows.

**Definition 1** *For any complexity classes $\mathcal{C}$ and $\mathcal{D}$ such that $\mathcal{C} \subseteq \mathcal{D}$, and for any oracle set $X$, we say that $\mathcal{C}^X = \mathcal{D}^X$* under stringent oracle access *if for any $\mathcal{D}$-machine $\mathcal{Q}_0$ whose query size is bounded by $q$, we can show some $\mathcal{C}$-machine $\mathcal{Q}_1$ such that for every input $x$, $\mathcal{Q}_1^X(x) = \mathcal{Q}_0^X(x)$, and $\mathcal{Q}_1^X(x)$ uses only queries of length $\leq q(|x|)$.*

## 2 Main Technical Result

We show the following collapsing result. $\mathrm{BPP}^C = \mathrm{NP}^C$ is its special case where $d = 1$.

**Theorem 1** *For any $d \geq 1$, we have some oracle $C$ such that $\mathrm{BPP}^C = \Sigma_d^{\mathrm{p},C}$ under stringent oracle access.*

Consider any oracle $\Sigma_d^{\mathrm{p}}$-machine $\mathcal{Q}_0$ for a canonical complete set for $\Sigma_d^{\mathrm{p},()}$. For the theorem, it suffices to show some oracle set $C$ and oracle BPP-machine $\mathcal{Q}_1$ that simulates $\mathcal{Q}_0^C$ under stringent oracle access.

Consider any input length $n$. We assume that the length of query of $\mathcal{Q}_0$ on any input of length $n$ is bounded by $q(n)$ with some polynomial. For simplifying our notation, assume that $q(n)$ also bounds the total computation time of $\mathcal{Q}_0$. Below we explain how to define our set $C$ so that some simple BPP-machine $\mathcal{Q}_1^C$ can simulate $\mathcal{Q}_0^C$ for every input $x \in \{0,1\}^n$. We design $\mathcal{Q}_1$ so that it asks only queries in $\{0,1\}^{q(n)}$ for such simulation. That is, the domain of inputs is $\{0,1\}^n$, and that of queries is $\{0,1\}^{q(n)}$. (For simplifying our discussion, we assume that $q(n)$ is large enough, say, $q(n) > n^3$.)

Let $\chi$ be the characteristic function of $C$ on $\{0,1\}^{q(n)}$; that is, for any $y \in \{0,1\}^{q(n)}$, $\chi(y) = 1$ if $y \in C$, and 0 otherwise. Instead of $C$, we will argue by using this $\chi$. Our goal is to define $\chi$ appropriately. Initially, we set $\chi(y) = *$ (i.e., undefined) for all $y \in \{0,1\}^{q(n)}$.

*Computer Sciences Dept., University of Wisconsin
†東京工業大学 情報理工学研究科 数理・計算科学専攻

For each input $x \in \{0,1\}^n$, let $Y_x$ be the set of query strings $y \in \{0,1\}^{q(n)}$ of the form $y = xw$ (hence, $|w| = q(n)-n$). Our idea is to encode the output $\mathcal{Q}_0^C(x)$ by using the value of $\chi$ on $y \in Y_x$. For example, it would be nice if we can set $\chi(y) = 1$ for all $y \in Y_x$ if and only if $\mathcal{Q}_0^C(x) = 1$. But the situation is not so easy because the oracle $C$ itself is determined by $\chi$. Thus, before encoding, we first fix the output $\mathcal{Q}_0^C(x)$ for all inputs $x \in \{0,1\}^n$. It turned out that we can fix $\mathcal{Q}_0^C(x)$ for all inputs $x \in \{0,1\}^n$ by setting the value of $\chi$ only on a certain subset $R \cup T$ of $\{0,1\}^{q(n)}$, and there are still many elements in $Y_x - (R \cup T)$. Note that the outputs are already fixed, we may define $\chi$ as we like on the remaining strings $y$ in $Y_x - (R \cup T)$; hence, we set $\chi(y) = \mathcal{Q}_0^C(x)$ for all such $y$. Furthermore, it can be shown that the value of $\chi$ on the $R \cup T$ part is mostly random; thus, $\chi(y) = \mathcal{Q}_0^C(x)$ for more than $1/2 + 1/\mathrm{poly}$ of $Y_x$. Therefore, by checking $\chi(y)$ (or, the membership of $y$ to $C$) for enough number of randomly chosen $y$ of $Y_x$, we would get the information $\mathcal{Q}_0^C(x)$ with high confidence. This is our BPP-machine $\mathcal{Q}_1$.

Now let us examine some of the technical points of our argument in more detail.

Following the argument in [2], we can regard the computation of $\mathcal{Q}_0$ on any fixed input $x$ with an *undetermined* oracle $X$ as a bounded depth Boolean circuit $C_x$ of the following type: The inputs are $2^{q(n)}$ Boolean variables $z_y$ ($y \in \{0,1\}^{q(n)}$), representing membership of a string $y \in \{0,1\}^{q(n)}$ in the oracle $X$. The Boolean circuit $C_x$ starts with an OR gate at the top, and alternate with AND's and OR's with depth $d+1$, where the bottom level gates have bounded fan-in at most $q(n)$, and all other AND and OR gates are unbounded fan-in. The overall circuit size is bounded by $q(n)2^{q(n)}$.

Note that what is undetermined is the oracle $X$, and $C_x$ is a circuit computing $\mathcal{Q}_0^X(x)$ for a given oracle $X$. In other words, when $X$ is fixed, the value is assigned to each input $z_y$ so that $z_y$ is 1 if and only if $y$ is in the oracle $X$; then one can compute the value of the circuit $C_x$.

We want to fix the value of the circuit by fixing the value of relatively small number of variables $z_y$. Our idea is to use *random restrictions*. We use $Z$ to denote the set of all variables $z_y$ of $C_x$. A *restriction* is a mapping from $Z$ to $\{0,1,*\}$, which we use to determine the value of each variable $z_y \in Z$. For any restriction $\rho$, let $C_x|\rho$ be a circuit obtained by assigning $\rho(z_y)$ to each variable $z_y$ of $C_x$. That is, $z_y$ becomes a constant 0 (resp., 1) if $\rho(z_y) = 0$ (resp., $\rho(z_y) = 1$), while $z_y$ remains as a variable if $\rho(z_y) = *$. A *random restriction* is to define each $\rho(z_y)$ independently at random with $\Pr\{\rho(z_y) = *\} = p$, and $\Pr\{\rho(z_y) = 0\} = \Pr\{\rho(z_y) = 0\} = (1-p)/2$, for some parameter $p$. It has been known (see, e.g., [2]) that a constant depth circuit can be greatly simplified by a random restriction with some suitably chosen $p$. We will use this result.

Results of this type are generally known as Switching Lemma. For our purpose, however, a slightly different version of Switching Lemma is helpful. We claim that by a random restriction, $C_x|\rho$ gets simplified so that it can be computed by a small depth decision tree. An important feature of small, say, depth $t$ decision tree is that we can determine its value by fixing values of some $t$ variables. The following decision tree version of Switching Lemma is due to Cai [1].

**Lemma 1** *For any depth $d+1$ Boolean circuit $C$ of size $\leq s$, and for any $t$ that bounds bottom fan-in, by a random restriction $\rho$ with $p = (10t)^{-d}$, we have $\Pr\{\, DC(C|\rho) \geq t \,\} \leq s/2^t$, where $DC(C|\rho)$ is the smallest depth of a a decision tree computing $C|\rho$.*

Let us apply this lemma to our circuit $C_x$. Recall that $s$, the circuit size bound, is $q(n)2^{q(n)}$. Though the bottom fan-in is bounded by $q(n)$, we set $t = 2q(n)$ so that $s/2^t = q(n)/2^{q(n)}$ For this $t$, we call a restriction $\rho$ *bad* if $\mathrm{DC}(C_x|\rho) \geq t$. Then, from the lemma, the probability of having a bad restriction is less than $q(n)/2^{q(n)}$. That is, with negligible probability, $C_x|\rho$ is expressed as a decision tree of depth $< t$, and the value of $C_x|\rho$ is determined by assigning values to less than $t$ variables.

So far, we have argued by considering any fixed input $x$ and the corresponding circuit $C_x$. Let us now consider the set of circuits $C_x$ for all $x \in \{0,1\}^n$. Although there are $2^n$ circuits, since $q(n)/2^{q(n)}$ is much smaller, the probability that we have a restriction that is bad for *some* $x$ is still exponentially small. Thus, after applying any "typical" random restriction $\rho$ to $C_x$, we need at most $t2^n$ additional assignments, which we call *decision tree assignments*, to determine the value of all circuits $C_x$. That is, by the corresponding partially defined set $\widehat{C}$, the output $\mathcal{Q}_0^C(x)$ is fixed for all $x \in \{0,1\}^n$.

Consider any $x \in \{0,1\}^n$ again, and we analyze the number of strings $y$ in $Y_x$ for which the value $\chi(y)$ is already fixed by a random restriction and decision tree assignments. Consider any typical random restriction $\rho$. Let $R$ (resp., $T$) be the set of query strings $y$ such that the corresponding $z_y$ is assigned 0 or 1 by $\rho$ (resp., by the decision tree assignments). Since $\|Y_x\| = 2^{q(n)-n}$, and since $\Pr\{\rho(z_y) = *\} = p = (20q(n))^{-d}$, we should have $\|Y_x - R\| \geq 2^{q(n)-n}/2(20q(n))^d$ for a "typical" $\rho$. Then we have $\|Y_x - (R \cup T)\| \geq 2^{q(n)-n}/4(20q(n))^d$, because $\|T\| \leq t2^n = q(n)2^{n+1} << 2^{q(n)-n}/4(20q(n))^d$. Thus, there are still a good number of strings $y$ in $Y_x - (R \cup T)$ for which $\chi$ is not fixed. Furthermore, since $\chi(y)$ is set 0 or 1 randomly on $R$, and $\|T\|$ is negligible compared with $\|Y_x - (R \cup T)\|$, we may assume that $\chi(y)$ is set 0 and 1 almost equally on $Y_x \cap (R \cup T)$. This is what we wanted to construct the oracle set $C$ and the BPP-machine $\mathcal{Q}_1$.

## References


[1] 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.

[2] M. Furst, J. Saxe, and M. Sipser, Parity, circuits, and the polynomial time hierarchy, in *Proc. 22nd IEEE Sympos. on Foundations of Comp. Sci.*, 260–270, 1981.