Electronic Colloquium on Computational Complexity, Report No. 34 (2000)

Efficiently Approximable Real-Valued Functions

Valentine Kabanets Charles Rackoff Stephen A. Cook
Department of Computer Science
University of Toronto
Toronto, Canada
{kabanets,rackoff,sacook}@cs.toronto.edu

April 25, 2000

Abstract

We consider a class, denoted APP, of real-valued functions $f : \{0,1\}^n \to [0,1]$ such that f can be approximated, to within any $\epsilon > 0$, by a probabilistic Turing machine running in time $\mathsf{poly}(n, 1/\epsilon)$. We argue that APP can be viewed as a generalization of BPP, and show that APP contains a natural complete problem: computing the acceptance probability of a given Boolean circuit; in contrast, no complete problems are known for BPP. We observe that all known complexity-theoretic assumptions under which BPP is easy (i.e., can be efficiently derandomized) imply that APP is easy; on the other hand, we show that BPP may be easy while APP is not, by constructing an appropriate oracle.

1 Introduction

The complexity class BPP is traditionally considered a class of languages that can be efficiently decided with the help of randomness. While it does contain some natural problems, the "semantic" nature of its definition (on every input, a BPP machine must have either at least 3/4 or at most 1/4 of accepting computation paths) makes it quite difficult to study as a class. One of the main obstacles for such a study is the lack of known complete problems for BPP. In fact, it is conjectured that BPP does not have any complete problems; some evidence towards the truth of this conjecture is provided by oracle constructions where BPP (and some other "semantic" complexity classes such as $\mathsf{NP} \cap \mathsf{co\text{-}NP}$, RP, and UP) do not have complete sets [Sip82, HH86] (see also [HI85, HLY86]).

Despite the difficulties, many researchers are trying to resolve various open questions regarding BPP, the most important one being whether BPP = P. All currently known approaches towards proving that BPP = P involve, explicitly or implicitly, constructing a polynomial-time algorithm for approximating the acceptance probability of any given Boolean circuit to within a small additive term ϵ, say $\epsilon < 1/4$. The existence of such an algorithm would clearly suffice to show that BPP = P. Indeed, for every input x to a BPP machine M, the acceptance probability of $M(x)$ is either at least 3/4 or at most 1/4, and we can easily construct a Boolean circuit $C_{M,x}$ simulating $M(x)$ so that $C_{M,x}$ has the same acceptance probability as $M(x)$; here the inputs to $C_{M,x}$ are random bits used by M on the input x.

Thus, on the one hand, there is no known complete problem for BPP, while, on the other hand, all efforts at showing that BPP is easy aim to show that the acceptance probability of any given Boolean circuit can be efficiently approximated to within a small additive factor. Below, we define a complexity class that is a natural generalization of BPP and prove that computing the acceptance probability of a given Boolean circuit is a complete problem for our class.

ISSN 1433-8092

To generalize BPP, we observe that a language L accepted by a BPP machine M is defined using an easily computable property of the acceptance probability of $M(x)$: $x \in L$ iff $M(x)$ accepts on at least $3/4$ of random strings; also note that the acceptance probability of a Boolean circuit C is easy to approximate to within any $\epsilon > 0$ in probabilistic time $\mathsf{poly}(|C|, 1/\epsilon)$ by random sampling. This suggests a definition of the class of real-valued functions $f : \{0,1\}^n \to [0,1]$ such that the value $f(x)$, for any $x \in \{0,1\}^n$, can be approximated to within any $\epsilon > 0$ in probabilistic time $\mathsf{poly}(n, 1/\epsilon)$. The class of such real-valued functions Approximable in Probabilistic Polynomial time will be denoted as APP, while the corresponding deterministic class as AP. It is worth pointing out that our definition of APP is different from the definition of the class of efficiently computable real functions f in [KF82, Ko91], where f is required to be approximable to within ϵ in probabilistic time $\mathsf{poly}(n, 1/\log\epsilon)$.

It is not hard to see that BPP corresponds exactly to the subclass of *Boolean* functions of APP. It is also easy to see that the problem of computing the acceptance probability of a given Boolean circuit is in APP. We will prove that, for an appropriate notion of reducibility, this problem is, in fact, complete for APP. Since all known complexity assumptions that imply $\mathsf{BPP} = \mathsf{P}$ actually imply that approximating the acceptance probability of a Boolean circuit is easy, our completeness result shows that every such complexity assumption implies that APP can be efficiently derandomized, i.e., that $\mathsf{APP} = \mathsf{AP}$.

It is interesting to consider the effects on the real world if $\mathsf{BPP} = \mathsf{P}$ on the one hand, and if $\mathsf{APP} = \mathsf{AP}$ on the other hand. If it turns out that $\mathsf{BPP} = \mathsf{P}$, then the effect would be minimal since very little of the real-world use of the randomness is in solving BPP problems. However, the effect of $\mathsf{APP} = \mathsf{AP}$, at least if this holds in an efficient way, would be quite pronounced. It appears that in this case, all the common uses of randomness in scientific computation could be removed (with only a polynomial loss of efficiency). The reason is that most uses of randomness in scientific computation are in probabilistic simulations; the goal of such a simulation is to estimate the probability of a certain event, or to estimate the value of a certain random variable. It is easy to see that if $\mathsf{APP} = \mathsf{AP}$, then such programs can be made deterministic.

A natural question is how these two assumptions, $\mathsf{BPP} = \mathsf{P}$ and $\mathsf{APP} = \mathsf{AP}$, are related to each other. Obviously, if $\mathsf{APP} = \mathsf{AP}$, then $\mathsf{BPP} = \mathsf{P}$. Is the converse true? Probably not. At least, such an implication is not provable by relativizable techniques: we construct an oracle where $\mathsf{BPP} = \mathsf{P}$ but approximating the acceptance probability of a Boolean circuit to within $1/4$ cannot be done in deterministic polynomial time.

Finally, we consider the question whether our two classes are recursively enumerable (r.e.). We show that AP is r.e., but leave it open whether the same holds for APP; it appears that APP may not be r.e., even though it contains a complete problem.

The remainder of the paper. In Section 2, we give formal definitions of our classes of efficiently approximable real-valued functions, APP and AP, and prove some of their properties. In Section 3, we exhibit the problems complete for these classes. We discuss the problem of derandomizing APP in Section 4. In Section 5, we construct an oracle world where $\mathsf{BPP} = \mathsf{P}$ but $\mathsf{APP} \neq \mathsf{AP}$. In Section 6, we show that the class AP is recursively enumerable. We conclude with Section 7.

2 Preliminaries

2.1 Definitions and Basic Properties of AP and APP

The class P can be thought of as the class of all polynomial-time computable uniform families $f = \{f_n\}_{n\geqslant 0}$ of Boolean functions $f_n : \{0,1\}^n \to \{0,1\}$; i.e., $f \in \mathsf{P}$ iff there is a polynomial-time Turing machine M such that, for all $n \in \mathbb{N}$, we have $M(x_1,\ldots,x_n) = f_n(x_1,\ldots,x_n)$. Similarly, BPP is the class of all uniform families of Boolean functions computable in probabilistic polynomial time. We would like to define a more general class of (not necessarily Boolean) function families. Let us consider real-valued functions $g_n : \{0,1\}^n \to [0,1]$. Since we cannot represent real numbers in the standard Turing-machine model of computation, we can try to approximate the real values of g_n by rational numbers. The following definitions are then natural.

Definition 1. We say that a deterministic Turing machine M *approximates* a family $g = \{g_n\}_{n\geqslant 0}$ of real-valued functions $g_n : \{0,1\}^n \to [0,1]$ if, for all $k, n \in \mathbb{N}$, we have $|M(1^k, x) - g_n(x)| \leqslant 1/k$. We call such a family g *polynomial-time approximable* if g is approximable by a polynomial-time Turing machine. Let AP (for *Approximable in Polynomial time*) consist of all polynomial-time approximable families g of real-valued functions $g_n : \{0,1\}^n \to [0,1]$.

The class of Boolean functions $f_n : \{0,1\}^n \to \{0,1\}$ can be viewed as a subclass of the class of all $[0,1]$-valued functions. Clearly, we have $\mathsf{P} \subset \mathsf{AP}$. In fact, the following is true.

Theorem 2. *The class* P *is exactly the subset of all Boolean function families in* AP.

Proof. As we observed above, P is contained in the subset of all Boolean function families in AP. For the other direction, consider any Boolean function family $g = \{g_n\}_{n\geqslant 0}$ in AP. By definition, there is a polynomial-time Turing machine M that can approximate $g_n(x)$ to within $1/k$ for any $k \in \mathbb{N}$. Let us choose $k = 3$. Then we can use M to distinguish between the case where $g_n(x) = 0$ and the case where $g_n(x) = 1$, and hence we can compute $g_n(x)$ exactly. □

Now we define the probabilistic analogue of AP, denoted by APP (for *Approximable in Probabilistic Polynomial time*), as the class of all families g of real-valued functions $g_n : \{0,1\}^n \to [0,1]$ that are approximable in polynomial time by *probabilistic* Turing machines. More precisely, we have the following.

Definition 3. A family $g = \{g_n\}_{n\geqslant 0}$ of real-valued function $g_n : \{0,1\}^n \to [0,1]$ is in APP if there exists a probabilistic polynomial-time Turing machine M such that, for all $k, n \in \mathbb{N}$, we have $\mathbf{Pr}[|M(1^k, x) - g_n(x)| \leqslant 1/k] \geqslant 3/4$, where the probability is taken over all random bits used by M.

Remark 4. The constant $3/4$ in the above definition is quite arbitrary, and, similarly to the case of BPP, it can be replaced by any $p_{k,n} \geqslant 1/2 + 1/\mathsf{poly}(k+n)$, or $p_{k,n} \leqslant 1 - 2^{-\mathsf{poly}(k+n)}$ (see Theorem 12).

We have the following analogue of Theorem 2 for BPP and APP.

Theorem 5. *The class* BPP *is exactly the subset of all Boolean function families in* APP.

Proof. The proof is very similar to that of Theorem 2. □

An important question is whether APP = AP; this is analogous to the long-standing open question whether BPP = P. The following easy theorem relates the two questions.

Theorem 6. *If* $\mathsf{APP} = \mathsf{AP}$, *then* $\mathsf{BPP} = \mathsf{P}$.

Proof. If $\mathsf{APP} \subseteq \mathsf{AP}$, then $\mathsf{BPP} \subseteq \mathsf{AP}$. Since BPP is a class of Boolean function families, we conclude by Theorem 2 that $\mathsf{BPP} \subseteq \mathsf{P}$. □

Remark 7. Theorem 20 below gives evidence against the converse of Theorem 6.

We would like to talk about hard problems in AP and APP. To this end, we define appropriate reductions.

Definition 8. We say that a family f of real-valued functions $f_n : \{0,1\}^n \to [0,1]$ is *polynomial-time many-one reducible* to a family g of real-valued functions $g_n : \{0,1\}^n \to [0,1]$, denoted $f \leqslant_m^p g$, if there is a polynomial-time computable family r of transformations $r_n : \{0,1\}^n \to \{0,1\}^m$, for some $m = m(n) \in \mathsf{poly}(n)$, such that, for all $n \in \mathbb{N}$,

$$f_n(x) = g_m(r_n(x)).$$

We say that f is *polynomial-time many-one approximately reducible* to g, denoted $f \lesssim_m^p g$, if there is a polynomial-time computable family r of transformations $r_{n,k} : \{1\}^k \times \{0,1\}^n \to \{0,1\}^m$, for some $m = m(n,k) \in \mathsf{poly}(n,k)$, such that, for all $k, n \in \mathbb{N}$,

$$|f_n(x) - g_m(r_{n,k}(1^k, x))| \leqslant 1/k.$$

Theorem 9. *Let* $f = \{f_n\}_{n\geqslant 0}$ *and* $g = \{g_n\}_{n\geqslant 0}$ *be any Boolean function families. Then* $f \leqslant_m^p g$ *iff* $f \lesssim_m^p g$.

Proof. Clearly, if $f \leqslant_m^p g$, then $f \lesssim_m^p g$. For the converse, assume that $f \lesssim_m^p g$ via a family of transformations $r_{n,k}(1^k, x_1, \ldots, x_n)$. It is easy to see that the family of transformations $r'_n(x_1, \ldots, x_n) = r_{n,2}(1^2, x_1, \ldots, x_n)$ will define a polynomial-time many-one reduction from f to g. □

It is also easy to see that AP and APP are closed under the two kinds of reductions defined above.

Theorem 10. *Let* f *and* g *be any* $[0,1]$*-valued function families such that* $f \lesssim_m^p g$. *We have the following implications:*

1. *if* $g \in \mathsf{AP}$, *then* $f \in \mathsf{AP}$; *and*

2. *if* $g \in \mathsf{APP}$, *then* $f \in \mathsf{APP}$.

Proof. Straightforward. □

2.2 Reducing the Error Probability for APP-Functions

Here we will justify Remark 4 that the class APP is robust with respect to the allowed probability of error. First, we recall the following Chernoff bound (see, e.g., [MR95] for a proof).

Lemma 11 (Chernoff). *Let* $X_1, X_2, \ldots, X_n$ *be independent Bernoulli variables with* $\mathbf{Pr}[X_i = 1] = p_i$, *let* $X = \sum_{i=1}^n X_i$, *and let* $\mu = \mathbf{E}[X]$. *Then, for any* $\delta \in (0,1]$, *we have* $\mathbf{Pr}[X < (1-\delta)\mu] < e^{-\mu\delta^2/2}$.

Now we can prove the following.

Theorem 12. *Let $g = \{g_n\}_{n \geqslant 0}$ be any family of real-valued function $g_n : \{0,1\}^n \to [0,1]$ for which there exist a probabilistic polynomial-time Turing machine M and a constant $c \in \mathbb{N}$ such that, for all $k, n \in \mathbb{N}$,*

$$\mathbf{Pr}[|M(1^k, x) - g_n(x)| \leqslant 1/k] \geqslant 1/2 + (k+n)^{-c}, \tag{1}$$

where the probability is taken over all random strings used by M. Then, for any $d \in \mathbb{N}$, there is a probabilistic polynomial-time Turing machine M' such that, for all $k, n \in \mathbb{N}$,

$$\mathbf{Pr}[|M'(1^k, x) - g_n(x)| \leqslant 1/k] \geqslant 1 - 2^{-(k+n)^d},$$

where the probability is taken over all random strings used by M'.

Proof. Fix any $k \in \mathbb{N}$ and $x \in \{0,1\}^n$. Let $N = k+n$ and let $\epsilon = 1/(3k)$. Consider an experiment in which $M(1^{3k}, x)$ is run m times, using independent random strings. Let $\alpha_1, \dots, \alpha_m$ be the outputs of $M(1^{3k}, x)$ in the corresponding runs. We define random variables X_i, $1 \leqslant i \leqslant m$, as follows: $X_i = 1$ if $|\alpha_i - g_n(x)| \leqslant \epsilon$, and $X_i = 0$ otherwise. Obviously, $X_1, \dots, X_m$ are independent Bernoulli variables such that, by (1), $\mathbf{Pr}[X_i = 1] = p \geqslant 1/2 + N^{-c}$, for all i. Let $X = \sum_{i=1}^m X_i$ and let $\mu = \mathbf{E}[X] = mp$.

The idea is to choose m large enough so that, with high probability, the majority of X_i's are equal to 1, i.e., the majority of α_i's are at most ϵ away from $g_n(x)$. Once we have more than a half of good α_i's, we will be able to find a value $\beta \in \mathbb{Q}$ which is at most $3\epsilon = 1/k$ away from $g_n(x)$. This will be the output of our new probabilistic machine $M'(1^k, x)$.

Now we provide the details. Let $\gamma = 2\epsilon$. We divide the interval $[0,1]$ into $\lceil 1/\gamma \rceil$ contiguous subintervals of length at most γ each. Let $0 = y_0, y_1, \dots, y_l = 1$ be the end points of these intervals, where $l = \lceil 1/\gamma \rceil$. We define the output of $M'(1^k, x)$ to be

$$\min_{1 \leqslant j \leqslant l} \{y_j \mid \text{the interval } [y_j - 2\epsilon, y_j + 2\epsilon] \cap [0,1] \text{ contains more than } m/2 \text{ of } \alpha_i\text{'s}\} \tag{2}$$

if such a j exists, and 0 otherwise.

An easy calculation shows that, for $\delta = N^{-c}$, we get $(1-\delta)\mu \geqslant m(1/2 + 1/(3N^c))$. By Lemma 11, choosing $m \geqslant 4N^{2c+d}$ guarantees that $X \geqslant (1-\delta)\mu$ with probability at least $1 - 2^{-N^d}$. Obviously, it suffices to prove the correctness of M' only for the case where $X \geqslant (1-\delta)\mu$. So, we assume below that at least $m(1/2 + 1/(3N^c)) > m/2$ of α_i's are good, i.e., the interval $I = [g_n(x) - \epsilon, g_n(x) + \epsilon] \cap [0,1]$ contains more than $m/2$ of α_i's.

Clearly, there is at least one point $y^* = y_{j^*}$ such that $|y^* - g_n(x)| \leqslant \gamma/2 = \epsilon$. Then, by our assumption above, the interval $[y^* - 2\epsilon, y^* + 2\epsilon] \cap [0,1]$ contains more than $m/2$ of α_i's, since it contains I as a subinterval. Hence, M' will always output the value satisfying Equation (2).

On the other hand, let $y = y_j$, for some $1 \leqslant j \leqslant l$, be such that the interval $J = [y - 2\epsilon, y + 2\epsilon] \cap [0,1]$ contains more than $m/2$ of α_i's. Then J must intersect I, since otherwise I would have fewer than $m/2$ of α_i's, contradicting our assumption above. Thus, we have $|y - g_n(x)| \leqslant 3\epsilon = 1/k$.

We conclude that, with probability at least $1 - 2^{-N^d}$, the output produced by $M'(1^k, x)$ will be at most $1/k$ away from $g_n(x)$. It should be also clear that M' runs in time $\mathsf{poly}(k, n)$. $\square$

2.3 Largeness of APP

In addition to the real-valued functions that result from the use of randomness in scientific computation, the class APP also contains all "properly normalized" #P functions. Recall that a family of functions $f_n : \{0,1\}^n \to \mathbb{N}$ is in #P if there is a polynomial-time computable relation $R(x, y)$, where $|y| = p(|x|)$ for some fixed polynomial p, such that $f_n(x) = \text{card}(\{y|R(x,y)\})$. We can normalize each such function family $f = \{f_n\}_{n\geqslant 0}$ by defining a new family $g = \{g_n\}_{n\geqslant 0}$ so that $g_n(x) = f_n(x)/2^{p(n)}$. Then g is in APP using a straightforward sampling algorithm, which can be analyzed using Chernoff's bound.

3 A Complete Function for APP

Below we exhibit a natural function family that is APP-complete under polynomial-time many-one approximate reductions. In contrast, no problem is known to be BPP-complete under polynomial-time many-one reductions, and hence, by Theorem 9, under polynomial-time many-one approximate reductions.

Circuit Acceptance Probability Problem (CAPP)
Given: A Boolean circuit C with n inputs.
Output: $\mathbf{Pr}_{x\in\{0,1\}^n}[C(x) = 1]$.

Clearly, CAPP gives rise to the well-defined family of $[0,1]$-valued functions. We will denote this function family by f_{CAPP}.

Theorem 13. *The function family f_{CAPP} is* APP*-complete under polynomial-time many-one approximate reductions.*

Proof. It is quite easy to see that f_{CAPP} is in APP, since we can approximate the acceptance probability of a given circuit by random sampling; the correctness of such an estimate will follow from appropriate Chernoff's bounds.

To prove that f_{CAPP} is APP-hard, we show that the problem of approximating any given real-valued function in APP can be reduced to CAPP. Let $g = \{g_n\}_{n\geqslant 0}$, where $g_n : \{0,1\}^n \to [0,1]$, be an arbitrary function family in APP, and let M be a probabilistic polynomial-time Turing machine that approximates g. Without loss of generality, we may assume that $M(1^k, x_1, \ldots, x_n)$ uses the same number $r(k,n) \in \mathsf{poly}(k+n)$ of random coin flips on every computation path, and that it produces an output $1/k$-close to $g_n(x)$ with probability at least $1 - 2^{-\mathsf{poly}(k+n)}$.

Given $x \in \{0,1\}^n$ and $k \in \mathbb{N}$, we will construct a Boolean circuit $C_{x,k}(w)$ such that $|g_n(x) - \mathbf{Pr}_w[C_{x,k}(w) = 1]| \leqslant 1/k$. To this end, we define a new, Boolean-valued, probabilistic machine M' such that $M'(1^k, x)$ first computes an output $\alpha \in [0,1]$ of $M(1^{2k}, x)$ by simulating M, and then accepts with probability α or rejects with probability $1-\alpha$.

We transform $M'(1^k, x)$ into a probabilistic circuit $C_{x,k}$. The crucial observation is that the acceptance probability of $C_{x,k}$ is exactly the average value of $M(1^{2k}, x)$ taken over all random strings of length $r(k,n)$. That is,

$$\mathbf{Pr}_{w\in\{0,1\}^{r(k,n)}}[C_{x,k}(w) = 1] = \mathbf{E}[M(1^{2k}, x)].$$

The theorem now follows by observing that $\mathbf{E}[M(1^{2k}, x)]$ is within $2/(2k) = 1/k$ of $g_n(x)$, provided that M computes a good output with probability at least $1 - 1/(4k)$. □

For any $\epsilon = \epsilon(n) > 0$, let us denote by ϵ-CAPP the problem of approximating, to within $\epsilon(n)$, the acceptance probability of a given n-input Boolean circuit. We have the following robustness result for CAPP.

Theorem 14. $f_{\text{CAPP}} \in$ AP *iff, for some $c \in \mathbb{N}$, $(1/2 - n^{-c})$-CAPP can be solved in deterministic polynomial time.*

Proof Sketch. $\Rightarrow$. Obvious.

$\Leftarrow$. Let C be a given n-input Boolean circuit, and let $\epsilon > 0$ be arbitrary. Consider $l+1$ points $p_0, p_1, \ldots, p_l$, where $l = \lceil 1/\epsilon \rceil$ and $p_0 = 0$, $p_i = i\epsilon$ for $0 < i < l$, and $p_l = 1$. We associate with each p_i a Boolean circuit D_i on kn inputs such that $D_i(x_1, \ldots, x_k) = 1$ iff $\left| \frac{\sum_{j=1}^{k} C(x_j)}{k} - p_i \right| \leqslant \epsilon$, where each $x_j \in \{0,1\}^n$. Let α_i be an approximation of the acceptance probability of each D_i to within $1/2 - (kn)^{-c}$, computable in deterministic polynomial time by assumption. Our algorithm will output the least p_i such that $\alpha_i > 1/2$.

By applying Chernoff's bounds, one can show that, for an appropriate $k \in \mathsf{poly}(|C|, 1/\epsilon)$, the described algorithm approximates the acceptance probability of C to within 2ϵ. □

4 APP vs. AP

Here we consider the problem of derandomizing the class APP. It appears that every known method for derandomizing the related class, BPP, yields a deterministic polynomial-time algorithm for approximating the APP-complete function f_{CAPP}. Hence, every such method for showing BPP = P can actually be used to show that APP = AP.

Indeed, all general methods for derandomizing BPP are constructive in the sense that they provide, under certain complexity assumptions, an efficient deterministic algorithm that decides the same language as a given BPP machine R. Moreover, such a deterministic algorithm on input x works by approximating the acceptance probability of a Boolean circuit that is obtained from $R(x)$; the inputs to this circuit are the random strings used by R on x.

All known methods for showing BPP = P are conditional on some complexity assumptions. Below we list several such assumptions, pointing where they have been shown to imply BPP = P.

*A*1. P = NP [Sip83].

*A*2. There is an $L \in \mathsf{DTIME}(2^{O(n)})$ that almost everywhere requires circuit size $2^{\Omega(n)}$ [IW97].

*A*3. There is a $\mathsf{poly}(n)$-time algorithm $\mathcal{D} : \{0,1\}^{O(\log n)} \rightarrow \{0,1\}^n$ (called a *Discrepancy Set Generator*) such that, for every Boolean circuit C_n of size n on n inputs, $|\mathbf{Pr}_{x\in\{0,1\}^n}[C_n(x) = 1] - \mathbf{Pr}_{y\in\{0,1\}^{O(\log n)}}[C_n(\mathcal{D}(y)) = 1]| < 1/4$ [folklore].

*A*4. There is a $\mathsf{poly}(n)$-time algorithm $\mathcal{H} : \{0,1\}^{O(\log n)} \rightarrow \{0,1\}^n$ (called a *Hitting Set Generator*) such that, for every Boolean circuit C_n of size n on n inputs, if $\mathbf{Pr}_{x\in\{0,1\}^n}[C_n(x) = 1] \geqslant 1/2$, then $C_n(\mathcal{H}(u)) = 1$ for at least one $u \in \{0,1\}^{O(\log n)}$ [ACR98].

*A*5. $f_{\mathrm{CAPP}} \in$ AP [folklore].

Theorem 15. $A1 \Rightarrow A2 \Leftrightarrow A3 \Leftrightarrow A4 \Rightarrow A5$.

Proof Sketch. $A1 \Rightarrow A2$. It is easy to see that $\mathsf{E}^{\Sigma_k^p}$, for some constant $k \in \mathbb{N}$, contains a language of maximum circuit complexity. If P = NP, then PH = P, and hence $\mathsf{E}^{\Sigma_k^p} \subseteq \mathsf{E}^{\mathsf{P}} \subseteq \mathsf{E}$.

$A2 \Rightarrow A3$. This is proved in [IW97].

$A3 \Rightarrow A4$. Trivial.

$A4 \Rightarrow A2$. See, e.g., [ISW99, Theorem 9].

$A4 \Rightarrow A5$. This is proved in [ACR98] (see also [ACRT99, BF99, GW99]). □

We would like to point out another complexity assumption that appears weaker than assumption $A4$ above, and yet it also implies $A5$. This assumption says that a *promise* version of Circuit SAT is easy.

$A6$. There is a deterministic polynomial-time algorithm that, given a Boolean circuit C on n inputs, outputs 0 if $\mathbf{Pr}_{x\in\{0,1\}^n}[C(x)=1]=0$, and 1 if $\mathbf{Pr}_{x\in\{0,1\}^n}[C(x)=1]\geqslant 1-2^{-\sqrt{n}}$.

Obviously, we have $A5 \Rightarrow A6$. More interestingly, the converse also holds.

Theorem 16. $A6 \Rightarrow A5$.

The proof of Theorem 16 can be extracted from [ACRT99], and is implicit in [BF99]. In the latter, a different assumption is used, namely that, for every probabilistic polynomial-time Turing machine M, there is a deterministic polynomial-time Turing machine A with the following property: for any input x, A accepts x if $\mathbf{Pr}[M \text{ accepts } x] \geqslant 1/2$, and A rejects x if $\mathbf{Pr}[M \text{ accepts } x]=0$. It is not hard to see that this assumption is equivalent to our assumption $A6$. For the sake of completeness, we prove Theorem 16 in Appendix A.

Since f_{CAPP} is APP-complete, we obtain the following.

Corollary 17. *If any of assumptions $A1$–$A6$ above is true, then* $\mathsf{APP}=\mathsf{AP}$.

We conclude this section with an easy theorem that demonstrates the "constructive nature" of the assumption $\mathsf{APP}=\mathsf{AP}$.

Theorem 18. *If* $\mathsf{APP}=\mathsf{AP}$*, then there is a deterministic polynomial-time Turing machine which, given a circuit C that accepts at least half of its inputs, outputs an input which C accepts.*

Proof Idea. We can find a good input to C, bit by bit, using an algorithm for CAPP to guide our search. Starting with the empty input, at each step we fix the next bit so that the acceptance probability of C over the remaining bits does not decrease too much. $\square$

As a corollary, if $\mathsf{APP}=\mathsf{AP}$, then any probabilistic polynomial-time algorithm for solving an NP search problem can be made deterministic.

5 APP vs. BPP

We know from Theorem 6 that if APP is easy, then so is BPP. Is the converse true? This question is particularly interesting since, as observed above, every known approach towards proving $\mathsf{BPP}=\mathsf{P}$ does yield $\mathsf{APP}=\mathsf{AP}$, and so it would be nice to prove that $\mathsf{BPP}=\mathsf{P}$ implies $\mathsf{APP}=\mathsf{AP}$.

Below we show that such an implication (if indeed true) cannot be proved using relativizable techniques: we construct an oracle A such that $\mathsf{BPP}^A=\mathsf{P}^A$, but $\mathsf{APP}^A\neq\mathsf{AP}^A$. Moreover, we show that, with respect to our oracle A, there is no polynomial-time algorithm for approximating, *to within* $1/4$, the acceptance probability of a given Boolean circuit. Thus, it is plausible that approximating the function f_{CAPP} may not be necessary for concluding that $\mathsf{BPP}=\mathsf{P}$, while it is necessary for concluding that $\mathsf{APP}=\mathsf{AP}$.

Remark 19. We observe that Theorems 15 and 16 relativize. Hence, with respect to the oracle A that we construct in this section, none of the assumptions $A1-A6$ holds, even though $\mathsf{BPP}^A=\mathsf{P}^A$.

The main theorem of this section is the following.

Theorem 20. *There is an oracle A such that* $\mathsf{BPP}^A = \mathsf{P}^A$, *but* $\mathsf{APP}^A \neq \mathsf{AP}^A$.

For the proof, we need the notion of an oracle circuit as defined in [Wil85]. An oracle circuit is a standard circuit which may also contain *oracle gates*, where a k-input oracle gate outputs 1 on input x if x belongs to the given oracle, and outputs 0 otherwise. The size of an oracle circuit is defined as the number of edges in its graph representation.

We define the relativized version of CAPP in the obvious way: for any oracle O, CAPP^O is the problem of computing the acceptance probability of a given oracle circuit relative to O. Let $[\epsilon\text{-CAPP}]^O$ denote the relativized version of ϵ-CAPP.

For an oracle O, we say that a probabilistic polynomial-time oracle machine R is a BPP *machine with respect to O* if the acceptance probability of R^O on any input $x \in \{0,1\}^*$ is either at most $1/4$ or at least $3/4$.

Theorem 21. *There is an oracle A such that* $\mathsf{BPP}^A = \mathsf{P}^A$, *but* $[1/4\text{-CAPP}]^A$ *is not solvable by any deterministic polynomial-time oracle Turing machine* D^A.

For our oracle construction, we use standard techniques, as found, e.g., in [BGS75, Rac82, Sip82, HH87, BI87, IN88, MV96]. A complete proof of Theorem 21 is given in Appendix B.

6 R.E. or Not R.E.

In this section, we consider the problem of recursively enumerating classes AP and APP. We say that AP is *recursively enumerable (r.e.)* if there is a deterministic Turing machine $M(x, 1^k, y)$ such that, for every fixed x, $M_x(1^k, y) = M(x, 1^k, y)$ runs in time $\mathsf{poly}(k, |y|)$, every M_x approximates some AP function, and every AP function is approximated by some M_x. The recursive enumerability of APP is defined similarly.

For any standard complexity class of languages, the existence of a complete problem under many-one polynomial-time reductions implies the recursive enumerability of this class: we can enumerate exactly all languages in such a class by enumerating all polynomial-time reductions. In contrast, it is not immediately clear whether the existence of complete functions for AP and APP should imply that these two classes are r.e. However, we can show that AP is r.e.

Theorem 22. *The class* AP *is r.e.*

Proof. We will describe a rational-valued computable function $F(x, y, 1^k)$ that is "universal" for the class of AP functions. That is, every x describes an AP function $f_x(y)$ approximated by $F_x(1^k, y) = F(x, y, 1^k)$, and, for every AP function $f(y)$, there is a string x such that $f(y)$ is approximated by $F_x(1^k, y)$.

Let the index x include a description of a Turing machine M, and a polynomial time bound $p(n, k)$ to be imposed on M. The actual machine computing $F_x(1^k, y)$ is a modification M' of M. Here is a recursive description of the action of M'. On input $(1^0, y)$, M' outputs $1/2$. On input $(1^k, y)$, where $k > 0$, M' simulates M on input $(1^{6k^2}, y)$ for the alloted time $p(n, 6k^2)$ and comes up with a tentative output r. If r is outside the interval $[0, 1]$, then M' outputs 0. Otherwise, M' recursively calls itself on input $(1^{k-1}, y)$, obtaining an output r'. If $|r - r'| < 1/k^2$, then M' outputs r; otherwise, M' outputs r'.

The machine M' is designed so that it runs in polynomial time, and for each fixed input y, there exists the limit of its outputs as k approaches infinity. Hence, M' is approximating a well-defined AP function f_x. On the other hand, our choice of parameters also ensures that, for every AP function $f(y)$ computable by a Turing machine M with a description x, the machine M' on input $(1^k, y)$ always outputs the value $M(1^{6k^2}, y)$, and hence $F_x(1^k, y)$ is approximating $f(y)$. $\square$

It is not clear, on the other hand, whether APP is r.e., even though it has a complete function.

7 Final Remarks

It is interesting to contemplate which uses of randomness would still be important, even if APP = AP. We would certainly still need randomness to choose keys in cryptographic applications. Randomness would also be necessary in order to "break symmetry" in many distributed computing applications. A third (less convincing) example is provided by some online algorithms, where we use randomness to defeat a certain class of offline adversaries. An especially interesting use of randomness is in the construction of certain combinatorial objects; if we have an efficient test for such an object, then "APP = AP" can be used for a deterministic construction (recall Theorem 18), but otherwise randomness seems necessary.

Lastly, we mention an example from science again. Consider a physicist who wants to do a probabilistic simulation; he doesn't know why, or what he wants to measure — he just wants to sit back and watch the simulation unfold, and look for something interesting. Unless we know how to model the physicist as a Turing Machine (and we don't), we will need randomness here as well. We invite the reader to think of other examples. (Hint: Anyone up for poker?)

References

[ACR98] A.E. Andreev, A.E.F. Clementi, and J.D.P. Rolim. A new general derandomization method. *Journal of the Association for Computing Machinery*, 45(1):179–213, 1998. (preliminary version in ICALP'96).

[ACRT99] A.E. Andreev, A.E.F. Clementi, J.D.P. Rolim, and L. Trevisan. Weak random sources, hitting sets, and BPP simulations. *SIAM Journal on Computing*, 28(6):2103–2116, 1999. (preliminary version in FOCS'97).

[BF99] H. Buhrman and L. Fortnow. One-sided versus two-sided error in probabilistic computation. In C. Meinel and S. Tison, editors, *Proceedings of the Sixteenth Annual Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 100–109. Springer Verlag, 1999.

[BGS75] T. Baker, J. Gill, and R. Solovay. Relativizations of the P=?NP question. *SIAM Journal on Computing*, 4(4):431–442, 1975.

[BI87] M. Blum and R. Impagliazzo. Generic oracles and oracle classes. In *Proceedings of the Twenty-Eighth Annual IEEE Symposium on Foundations of Computer Science*, pages 118–126, 1987.

[GW99] O. Goldreich and A. Wigderson. Improved derandomization of BPP using a hitting set generator. In D. Hochbaum, K. Jansen, J.D.P. Rolim, and A. Sinclair, editors, *Randomization, Approximation, and Combinatorial Optimization*, volume 1671 of *Lecture Notes in Computer Science*, pages 131–137. Springer Verlag, 1999. (RANDOM-APPROX'99).

[HH86] J. Hartmanis and L. Hemachandra. Complexity classes without machines: On complete languages for UP. In L. Kott, editor, *Proceedings of the Thirteenth International Colloquium on Automata, Languages, and Programming*, volume 226 of *Lecture Notes in Computer Science*, pages 123–135. Springer Verlag, 1986.

[HH87] J. Hartmanis and L.A. Hemachandra. One-way functions, robustness, and the non-isomorphism of NP-complete sets. In *Proceedings of the Second Annual IEEE Conference on Structure in Complexity Theory*, pages 160–174, 1987.

[HI85] J. Hartmanis and N. Immerman. On complete problems for NP$\cap$CoNP. In W. Brauer, editor, *Proceedings of the Twelfth International Colloquium on Automata, Languages, and Programming*, volume 194 of *Lecture Notes in Computer Science*, pages 250–259. Springer Verlag, 1985.

[HLY86] J. Hartmanis, M. Li, and Y. Yesha. Containment, separation, complete sets, and immunity of complexity classes. In L. Kott, editor, *Proceedings of the Thirteenth International Colloquium on Automata, Languages, and Programming*, volume 226 of *Lecture Notes in Computer Science*, pages 136–1145. Springer Verlag, 1986.

[IN88] R. Impagliazzo and M. Naor. Decision trees and downward closures. In *Proceedings of the Third Annual IEEE Conference on Structure in Complexity Theory*, pages 29–38, 1988.

[ISW99] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Near-optimal conversion of hardness into pseudo-randomness. In *Proceedings of the Fortieth Annual IEEE Symposium on Foundations of Computer Science*, pages 181–190, 1999.

[IW97] R. Impagliazzo and A. Wigderson. P=BPP if E requires exponential circuits: Derandomizing the XOR Lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 220–229, 1997.

[KF82] K. Ko and H. Friedman. Computational complexity of real functions. *Theoretical Computer Science*, 20:323–352, 1982.

[Ko91] K. Ko. *Complexity Theory of Real Functions.* Birkhäuser, Boston, 1991.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, New York, 1995.

[MV96] A.A. Muchnik and N.K. Vereshchagin. A general method to construct oracles realizing given relationships between complexity classes. *Theoretical Computer Science*, 157:227–258, 1996.

[Rac82] C. Rackoff. Relativized questions involving probabilistic algorithms. *Journal of the Association for Computing Machinery*, 29(1):261–268, 1982.

[Sip82] M. Sipser. On relativization and the existence of complete sets. In *Proceedings of the Ninth International Colloquium on Automata, Languages, and Programming*, volume 140 of *Lecture Notes in Computer Science*, pages 523–531. Springer Verlag, 1982.

[Sip83] M. Sipser. A complexity theoretic approach to randomness. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 330–335, 1983.

[Wil85] C.B. Wilson. Relativized circuit complexity. *Journal of Computer and System Sciences*, 31:169–181, 1985.

A Proof of Theorem 16

First, for every $\epsilon = \epsilon(n) > 0$, we define the following promise problem.

$\epsilon(n)$**-SAT**
Given: An n-input Boolean circuit C.
Output: 0 if $\mathbf{Pr}_{x\in\{0,1\}^n}[C(x) = 1] = 0$, and 1 if $\mathbf{Pr}_{x\in\{0,1\}^n}[C(x) = 1] \geqslant \epsilon(n)$.

The following lemma shows the robustness of ϵ-SAT; a version of this lemma for the case of Hitting Set Generators was proved in [ACR98].

Lemma 23. *If* $(1-2^{-\sqrt{n}})$*-SAT is solvable in deterministic polynomial time, then there is a deterministic polynomial-time algorithm* $\mathcal{A}$ *such that* $\mathcal{A}(1^k, C)$ *solves* $1/k$*-SAT for all circuits* C *and all* $k \in \mathbb{N}$.

Proof. We describe a reduction from $\epsilon(n)$-SAT to $(1-2^{-\sqrt{n}})$-SAT, which runs in time polynomial in the size of the input circuit and $1/\epsilon(n)$. For any given Boolean circuit $C : \{0,1\}^n \to \{0,1\}$ with $\mathbf{Pr}_{x\in\{0,1\}^n}[C(x) = 1] \geqslant \epsilon = \epsilon(n)$, define $C^t(x_1, \ldots, x_t) = \bigvee_{j=1}^{t} C(x_j)$, where each $x_j \in \{0,1\}^n$. Then $\mathbf{Pr}_{x\in\{0,1\}^{nt}}[C^t(x) = 0] \leqslant (1-\epsilon(n))^t \leqslant e^{-\epsilon t}$, which is at most $2^{-\sqrt{tn}}$ for $t \geqslant n/\epsilon^2$ and sufficiently large n. □

Thus, in our proof of Theorem 16, we can assume that we are given a deterministic algorithm $\mathcal{A}$ for solving $\epsilon(n)$-SAT on an n-input Boolean circuit C in time $\mathsf{poly}(|C|, 1/\epsilon(n))$.

We will need the notion of a *discrepant* set for a Boolean function (circuit).

Definition 24. For a Boolean n-input circuit C and any $0 < \epsilon < 1$, we say that a set $\varnothing \neq S \subseteq \{0,1\}^n$ is *ϵ-discrepant for C* if $|\mathbf{Pr}_{x\in\{0,1\}^n}[C(x) = 1] - \mathbf{Pr}_{y\in S}[C(y) = 1]| \leqslant \epsilon$.

For $S \subseteq \{0,1\}^n$ and $a \in \{0,1\}^n$, we define the set $S \oplus a = \{s \oplus a |\ s \in S\}$, where $s \oplus a$ is the result of XORing s and a bitwise; in other words, $S \oplus a$ is an affine translation of S by a in the Boolean vector space.

Definition 25. For a Boolean n-input circuit C and any $0 < \epsilon < 1$, we say that a set $\varnothing \neq S \subseteq \{0,1\}^n$ is *ϵ-concentrated with respect to C* if $|p_{\min}(C,S) - p_{\max}(C,S)| \leqslant \epsilon$, where $p_{\min}(C,S) = \min_{a\in\{0,1\}^n}\{\mathbf{Pr}_{x\in S\oplus a}[C(x) = 1]\}$ and $p_{\max}(C,S)$ is defined similarly.

An interesting relationship between discrepancy and concentration is provided by the following lemma, which is implicit in [ACR98, ACRT99].

Lemma 26. *For a Boolean n-input circuit C and any $0 < \epsilon < 1$, if a set $\varnothing \neq S \subseteq \{0,1\}^n$ is ϵ-concentrated with respect to C, then S is ϵ-discrepant for C.*

Proof. It is not difficult to see that, for any $S \neq \varnothing$, we have

$$\frac{1}{2^n} \sum_{a\in\{0,1\}^n} \mathbf{Pr}_{x\in S\oplus a}[C(x) = 1] = \mathbf{Pr}_{x\in\{0,1\}^n}[C(x) = 1]. \tag{3}$$

Since the average value of any function over a finite set is contained between its minimum and its maximum over this set, we conclude that both $\mathbf{Pr}_{x\in\{0,1\}^n}[C(x) = 1]$ and $\mathbf{Pr}_{x\in S}[C(x) = 1]$ are contained between $p_{\min}(C,S)$ and $p_{\max}(C,S)$, and hence, S is ϵ-discrepant for C. □

Next we show that, for any Boolean circuit C, there exist small sets that are concentrated with respect to C.

Lemma 27. *For any n-input Boolean circuit C and any $0 < \epsilon < 1$, there exists an $m \in \mathsf{poly}(n, 1/\epsilon)$ such that at least a half of all sets $S \subseteq \{0,1\}^n$ of size m are ϵ-concentrated with respect to C.*

Proof. As in [ACR98], we prove the following two claims, which will imply our lemma.

Claim 28. *For any n-input Boolean circuit C and any $0 < \epsilon < 1$, there exists an $m \in \mathsf{poly}(n, 1/\epsilon)$ such that, for at least a half of sets $S \subseteq \{0,1\}^n$ of size m, the following holds: $S \oplus a$ is ϵ-discrepant for C, for every $a \in \{0,1\}^n$.*

Proof of Claim 28. Fix any $a \in \{0,1\}^n$. Let S be a set of m vectors from $\{0,1\}^n$ chosen independently and uniformly at random. Clearly, the vectors in $S \oplus a$ are also uniformly distributed and independent.

By Chernoff's bound, the probability that $S \oplus a$ is not ϵ-discrepant for C can be made smaller than 2^{-n-1} by choosing $m \in \mathsf{poly}(n, 1/\epsilon)$. Thus, the probability that, for a random S of size m, there exists an $a \in \{0,1\}^n$ such that $S \oplus a$ is not ϵ-discrepant for C is smaller than $2^n 2^{-n-1} = 1/2$. $\square$

Claim 29. *For any n-input Boolean circuit C, set $\varnothing \neq S \subseteq \{0,1\}^n$, and $0 < \epsilon < 1$, if $S \oplus a$ is ϵ-discrepant for C for every $a \in \{0,1\}^n$, then S is 2ϵ-concentrated with respect to C.*

Proof of Claim 29. Let $a_{\min}, a_{\max} \in \{0,1\}^n$ be such that $p_{\min}(C, S) = \mathbf{Pr}_{x \in S \oplus a_{\min}}[C(x) = 1]$ and $p_{\max}(C, S) = \mathbf{Pr}_{x \in S \oplus a_{\max}}[C(x) = 1]$. Since $S \oplus a_{\min}$ is ϵ-discrepant for C, we get that

$$\begin{aligned} p_{\min}(C, S) &= \mathbf{Pr}_{x \in S \oplus a_{\min}}[C(x) = 1] \\ &\geqslant \mathbf{Pr}_{x \in \{0,1\}^n}[C(x) = 1] - \epsilon \end{aligned}$$

and, similarly, since $S \oplus a_{\max}$ is ϵ-discrepant for C, we get that $p_{\max}(C, S) \leqslant \mathbf{Pr}_{x \in \{0,1\}^n}[C(x) = 1] + \epsilon$. Hence, $p_{\max}(C, S) - p_{\min}(C, S) \leqslant 2\epsilon$. $\square$

This finishes the proof of Lemma 27. $\square$

For a given n-input Boolean circuit C and $0 < \epsilon < 1$, consider the algorithm D-Test$_{C,\epsilon}$ given in Figure 1. Note that the running time of D-Test$_{C,\epsilon}$ is polynomial in $|C|$ and $1/\epsilon$, assuming the existence of a polynomial-time algorithm $\mathcal{A}$ for promise SAT. We shall argue that every set S accepted by D-Test$_{C,\epsilon}$ is 2ϵ-discrepant for C, and that D-Test$_{C,\epsilon}$ accepts many sets S.

Theorem 30 (Soundness of Discrepancy Test). *If* D-Test$_{C,\epsilon}$ *accepts a set S, then S is 2ϵ-discrepant for C.*

Proof. Let S be any set accepted by D-Test$_{C,\epsilon}$. It follows that there exist $0 \leqslant p_1, p_2 \leqslant 1$ such that, for all but at most ϵ fraction of vectors $a \in \{0,1\}^n$, we have $p_1 \leqslant \mathbf{Pr}_{x \in S \oplus a}[C(x) = 1] \leqslant p_2$. Using Identity (3), we obtain that $p_1 - \epsilon \leqslant \mathbf{Pr}_{x \in \{0,1\}^n}[C(x) = 1] \leqslant p_2 + \epsilon$. The theorem follows. $\square$

Theorem 31 (Richness of Discrepancy Test). *At least a half of all input sets $S \subseteq \{0,1\}^n$ will be accepted by* D-Test$_{C,\epsilon}$.

Proof. It follows from Lemma 27 that at least a half of all sets S of size m are such that $p_{\max}(C, S) - p_{\min}(C, S) \leqslant \epsilon$. Since, for $p_1 = p_{\min}(C, S)$ and $p_2 = p_{\max}(C, S)$, $\mathbf{Pr}_{a \in \{0,1\}^n}[C^{\text{bad}}(a) = 1] = 0$, we conclude that each such set S will pass the test. $\square$

```
D-TEST$_{C,\epsilon}(S)$
Input: a set $S \subseteq \{0,1\}^n$ of size $m$, where $m$ is as in the proof of Lemma 27.
begin
  for all  $i, j : 0 \leqslant i \leqslant j \leqslant m$ do
    $p_1 := i/m$; $p_2 := j/m$;
    $C^{\mathrm{bad}}$ :=an $n$-input Boolean circuit such that $C^{\mathrm{bad}}(a) = 0$ iff
            $p_1 \leqslant \mathbf{Pr}_{x \in S \oplus a}[C(x) = 1] \leqslant p_2$
    if $(p_1 \leqslant \mathbf{Pr}_{x \in S}[C(x) = 1] \leqslant p_2)$ and $(p_2 - p_1 \leqslant \epsilon)$ and $(\mathcal{A}(1^{\lceil \epsilon^{-1} \rceil}, C^{\mathrm{bad}}) = 0)$
    then return 1
    end if
  end for
  return 0
end
```

Figure 1: Discrepancy test

To finish the proof of Theorem 16, we will show how to approximate the acceptance probability of a given Boolean circuit C, using the algorithm $\mathcal{A}$ for promise SAT and the algorithm D-TEST described above. Observe that each set S passing the discrepancy test yields a correct estimate $\tilde{p}(C, S)$ of the acceptance probability of C, where $\tilde{p}(C, S) = i/m$ for some $0 \leqslant i \leqslant m$. Let us map each set S passing the test to the value $\tilde{p}(C, S)$. Since there are at most $m + 1$ different values for $\tilde{p}(C, S)$, there is an i, $0 \leqslant i \leqslant m$, that gets at least $1/(2(m + 1))$ fraction of all sets S passing the discrepancy test, which is at least $1/(4(m + 1))$ of all sets S of size m by Theorem 31. This suggests the algorithm ESTIMATOR given in Figure 2.

```
ESTIMATOR$(C, \epsilon)$
Input: an $n$-input Boolean circuit $C$ and $0 < \epsilon < 1$.
begin
  for all  $i : 0 \leqslant i \leqslant m$ do
    $\bar{p} := i/m$;
    $C^{\text{d-test}}$ :=an $mn$-input Boolean circuit such that $C^{\text{d-test}}(S) = 1$ iff
            D-TEST$_{C,\epsilon}(S) = 1$ and $\bar{p} = \mathbf{Pr}_{x \in S}[C(x) = 1]$
    if $(\mathcal{A}(1^{\lceil (4(m+1))^{-1} \rceil}, C^{\text{d-test}}) = 1)$ then return $\bar{p}$
    end if
  end for
end
```

Figure 2: Estimator

The algorithm ESTIMATOR(C, ϵ) approximates the acceptance probability of C to within 2ϵ. Indeed, the soundness of ESTIMATOR follows from that of D-TEST (Theorem 30); its completeness follows from the existence of $\bar{p} = i/m$, $0 \leqslant i \leqslant m$, for which $C^{\text{d-test}}$ has the acceptance probability at least $1/(4(m + 1))$, and therefore it will be accepted by $\mathcal{A}$. It is easy to check that the running time of ESTIMATOR(C, ϵ) is polynomial in $|C|$ and $1/\epsilon$.

B Proof of Theorem 21

Define $S = \{n_i \mid i \in \mathbb{N}\}$ to be the set of lengths n_i, where $n_0 = 1$ and $n_{i+1} = 2^{2^{n_i}}$. Our oracle A will be of the form $B \oplus \mathrm{QBF}$, where QBF is the PSPACE-complete language of true quantified Boolean formulas; here, for sets B and C, $B \oplus C = \{0u \mid u \in B\} \cup \{1v \mid v \in C\}$. The set B will not contain any words of length $l \in \mathbb{N} \setminus S$.

Define c_n to be the oracle circuit on n inputs that accepts $x \in \{0,1\}^n$ iff $0x$ belongs to the oracle. Observe that $\mathbf{Pr}_{x\in\{0,1\}^n}[c_n^{B\oplus C}(x) = 1]$ is exactly the fraction of n-bit strings in B. Let $e_n = \mathrm{Enc}(c_n)$ be a binary encoding of the circuit c_n, using any fixed encoding function Enc. Let $D_0, D_1, D_2, \ldots$ and $R_0, R_1, R_2, \ldots$ be enumerations of deterministic and probabilistic polynomial-time oracle Turing machines, respectively.

Construction. We define the set B in stages; at each stage, B is defined for a finite set of lengths. First, B is undefined for all lengths. At stage $i = 2k+1$, we extend the definition of B so that D_k fails to solve 1/4-CAPP with respect to this extension. At stage $i = 2k+2$, we extend, if possible, the definition of B so that R_k is *not* a BPP machine with respect to this extension. Note that, everywhere below, whenever we talk about an extension of an oracle defined for finitely many lengths to a completely defined oracle, we consider only those extensions that are empty for all lengths $l \in \mathbb{N} \setminus S$. In more detail, the construction of B is given in Figure 3.

Stage 0: B_0 is undefined for all lengths $l \in \mathbb{N}$.
Stage i: Let B_{i-1} be defined for all lengths $\leqslant m$.
`Case` $i = 2k+1$: Choose the smallest $n \in S$ such that $n > m$ and D_k on input e_n runs in time less than 2^{n-1}. Let C be the extension of B_{i-1} that is empty on all lengths $> m$, and let $D_k^{C\oplus\mathrm{QBF}}(e_n) = \alpha \in \mathbb{Q}$. Denote by Y the set of all oracle queries made by $D_k(e_n)$ to C; by the above, $|Y| < 2^{n-1}$. There are two subcases.

1. $\alpha > 1/4$. Then extend the definition of B_{i-1} so that B_i is empty on lengths $m < l \leqslant n$.
2. $\alpha \leqslant 1/4$. Then extend the definition of B_{i-1} by adding any $2^{n-1}+1$ strings of length n that are not in Y; make B_i empty on all lengths $m < l < n$.

`Case` $i = 2k+2$: There are two subcases.

1. There is an extension C of B_{i-1} such that $R_k^{C\oplus\mathrm{QBF}}$ is not a BPP machine with respect to $C \oplus \mathrm{QBF}$, i.e., there is an input $x \in \{0,1\}^*$ such that $\mathbf{Pr}[R_k^{C\oplus\mathrm{QBF}}(x) = 1] \in (1/4; 3/4)$. Let Y be the set of all queries made by $R_k(x)$ to C, over all random choices of R_k, and let $n = \max_{y\in Y} |y|$. Then extend the definition of B_{i-1} to all lengths $m < l \leqslant n$ so that $y \in B_i$ iff $y \in C$, for all $y \in Y$.
2. For every extension C of B_{i-1}, R_k is a BPP machine with respect to $C \oplus \mathrm{QBF}$. Then leave the definition of B_i the same as that of B_{i-1}.

Figure 3: Defining the set B.

Correctness. The correctness of our construction follows from the next two claims.

Claim 32. $[1/4\text{-CAPP}]^{B\oplus\mathrm{QBF}}$ *cannot be solved by any deterministic polynomial-time machine using* $B \oplus \mathrm{QBF}$ *as an oracle.*

Proof of Claim 32. Note that, at stage $i = 2k+1$, the kth polynomial-time machine D_k is defeated:

the set B_i is defined so that $D_k^{B\oplus\mathrm{QBF}}(e_n) = \alpha$, where α is more than 1/4 away from the correct answer for e_n with respect to B. □

Claim 33. $\mathsf{BPP}^{B\oplus\mathrm{QBF}} = \mathsf{P}^{B\oplus\mathrm{QBF}}$.

Proof of Claim 33. Our proof is very similar to the proof of a related result in [MV96, Lemma 4.2].

Let $R = R_k$ be any probabilistic polynomial-time oracle machine that was not defeated at stage $i = 2k+2$ of our oracle construction, i.e., for any oracle C extending B_{i-1}, the machine R is a BPP machine with respect to $C \oplus \mathrm{QBF}$. It suffices to show that there is a deterministic oracle machine M such that

1. $M^B(x) = R^{B\oplus\mathrm{QBF}}(x)$ for all $x \in \{0,1\}^*$, and
2. on input x, M runs within $\mathsf{poly}(|x|)$ space and makes at most $\mathsf{poly}(|x|)$ oracle queries.

Using the PSPACE-completeness of QBF, such a machine M can then be easily converted into a deterministic polynomial-time oracle machine $\hat{M}^{B\oplus\mathrm{QBF}}$ accepting the same language as M^B (see, e.g., [MV96, Lemma 1.3] for a proof of this).

The algorithm for M is as follows. On input x, we first find the smallest $n = n_j \in S$ such that $\log_2 n \leqslant |x| < 2^n$. Without loss of generality, we may assume that x is sufficiently long so that R on x cannot query strings of length $\geqslant n_{j+1}$, and that n is bigger than the maximum length for which B_i was defined.

Let $t \in \mathsf{poly}(|x|)$ be the running time of R on input x, and let $r \leqslant t$ be the number of random bits used by R on any single computation path. For an oracle O, we associate with each of 2^r possible random strings α_l used by $R^{O\oplus\mathrm{QBF}}(x)$ the set Q_l of n-bit strings for which R queries O, when using α_l as a random string. For a string $y \in \{0,1\}^n$, we denote by $w_O(y)$ the fraction of Q_l's containing y. Clearly,

$$\sum_{y\in\{0,1\}^n} w_O(y) \leqslant t, \tag{4}$$

for any oracle O, since each of the 2^r random strings contributes at most t different y's. Let $W_O = \{y \in \{0,1\}^n |\ w_O(y) \geqslant 1/(8t)\}$. Inequality (4) immediately yields that $|W_O| \leqslant 8t^2$. The following lemma is an adaptation of a claim in [MV96, Claim, p. 240].

Lemma 34. *Let B' be any extension of B_{i-1}, and let C be any oracle that agrees with B' on all words from $W = W_{B'}$ and all words of length less than n.*

1. *If $\mathbf{Pr}[R^{B'\oplus\mathrm{QBF}}(x) = 1] \geqslant 3/4$, then $\mathbf{Pr}[R^{C\oplus\mathrm{QBF}}(x) = 1] \geqslant 3/4$.*
2. *If $\mathbf{Pr}[R^{B'\oplus\mathrm{QBF}}(x) = 1] \leqslant 1/4$, then $\mathbf{Pr}[R^{C\oplus\mathrm{QBF}}(x) = 1] \leqslant 1/4$.*

Proof of Lemma 34. `Statement 1`. Suppose that there is an oracle C that agrees with B' on all words from W and all words of length less than n, but $\mathbf{Pr}[R^{C\oplus\mathrm{QBF}}(x) = 1] < 3/4$. Since R is a BPP machine with respect to $C\oplus\mathrm{QBF}$, this implies that $\mathbf{Pr}[R^{C\oplus\mathrm{QBF}}(x) = 1] \leqslant 1/4$. Let us choose such an oracle C that differs from B' on the *smallest* number of strings. Denote by U the set of those n-bit strings where B' and C differ.

By a counting argument, we show that $w_C(y) \geqslant 1/2$ for each $y \in U$. Indeed, for any $y \in U$, let C_y be obtained from C by adding y to C if $y \notin C$, or removing y from C if $y \in C$. Since $U \cap W = \varnothing$, the set C_y still agrees with B' on all words from W and all words of length less than n, but it differs from B' on fewer words than C does. The minimality of C implies that

$\mathbf{Pr}[R^{C_y \oplus \mathrm{QBF}}(x) = 1] \geqslant 3/4$. Thus, changing C on a single string y resulted in a jump of at least $1/2$ in the acceptance probability of R on x, which is possible only if $w_C(y) \geqslant 1/2$.

Now, Inequality (4) implies that $|U| \leqslant 2t$. Since $U \cap W = \varnothing$, we get that $w_{B'}(y) < 1/(8t)$ for each $y \in U$, and so $\sum_{y \in U} w_{B'}(y) < 2t/(8t) = 1/4$. Thus, changing B' on strings in U can affect the acceptance probability of R on x by less than $1/4$, but $\mathbf{Pr}[R^{B' \oplus \mathrm{QBF}}(x) = 1] - \mathbf{Pr}[R^{C \oplus \mathrm{QBF}}(x) = 1] \geqslant 3/4 - 1/4 = 1/2$, which is a contradiction.

`Statement 2.` The proof is analogous to that of Statement 1 above. □

For any oracle O and any set $U \subseteq \{0,1\}^n$, we denote by $Q(O, U)$ the set of all oracles C such that C agrees with O on all words in U and all words of length less than n, and C contains at most $8t^2$ words from $\{0,1\}^n \setminus U$. As in [MV96], we obtain the following.

Lemma 35. *Let B' be any extension of B_{i-1}.*

1. $\mathbf{Pr}[R^{B' \oplus \mathrm{QBF}}(x) = 1] \geqslant 3/4$ *iff there is a set* $U \subseteq \{0,1\}^n$ *of size at most* $8t^2$ *such that* $\mathbf{Pr}[R^{C \oplus \mathrm{QBF}}(x) = 1] \geqslant 3/4$ *for every oracle* $C \in Q(B', U)$.

2. $\mathbf{Pr}[R^{B' \oplus \mathrm{QBF}}(x) = 1] \leqslant 1/4$ *iff there is a set* $U \subseteq \{0,1\}^n$ *of size at most* $8t^2$ *such that* $\mathbf{Pr}[R^{C \oplus \mathrm{QBF}}(x) = 1] \leqslant 1/4$ *for every oracle* $C \in Q(B', U)$.

Proof of Lemma 35. `Statement 1.` $\Rightarrow$. This follows from Statement 1 of Lemma 34 by choosing $U = W_{B'}$.

$\Leftarrow$. Suppose that there is a set $U \subseteq \{0,1\}^n$ of size at most $8t^2$ such that $\mathbf{Pr}[R^{C \oplus \mathrm{QBF}}(x) = 1] \geqslant 3/4$ for every oracle $C \in Q(B', U)$. Let $W = W_{B'}$, and let D be the oracle that agrees with B' on all words of length less than n and all words in $U \cup W$, and that contains no words from $\{0,1\}^n \setminus (U \cup W)$. Then $\mathbf{Pr}[R^{D \oplus \mathrm{QBF}}(x) = 1] \geqslant 3/4$, and hence, by Statement 2 of Lemma 34, we should have $\mathbf{Pr}[R^{B' \oplus \mathrm{QBF}}(x) = 1] \geqslant 3/4$.

`Statement 2.` The proof is analogous to that of Statement 1 above. □

Let B' be any extension of B_{i-1}. Observe that each set U containing at most $8t^2 \in \mathsf{poly}(|x|)$ binary strings can be specified with at most $\mathsf{poly}(|x|)$ bits. Also, for every such U, each oracle $C \in Q(B', U)$ can be completely specified with only $\mathsf{poly}(|x|)$ bits: there are at most $\mathsf{poly}(|x|)$ binary strings of length less than n, and C may contain at most $16t^2$ strings of length n. Hence, for such C's, we can compute $\mathbf{Pr}[R^{C \oplus \mathrm{QBF}}(x) = 1]$ in polynomial space, querying B' on at most $\mathsf{poly}(|x|)$ strings.

To conclude the proof, we use the techniques from [BI87, HH87]. Our polynomial-space algorithm for deciding the language of $R^{B \oplus \mathrm{QBF}}$ that uses at most $\mathsf{poly}(|x|)$ queries to B is sketched below.

On input x, repeat the next step for $l = 1, \ldots, 8t^2$. Find a set $U(l) \subseteq \{0,1\}^n$ of size at most $8t^2$ and an extension $B'(l)$ of B_{i-1} such that $B'(l)$ agrees with B on all strings queried so far, and $\mathbf{Pr}[R^{C \oplus \mathrm{QBF}}(x) = 1] \geqslant 3/4$ for every oracle $C \in Q(B'(l), U(l))$. Query B on all strings in $U(l)$. If no such $U(l)$ and $B'(l)$ can be found, then reject.

If x was not rejected in the loop above, try to find a set $U \subseteq \{0,1\}^n$ of size at most $8t^2$ such that B was already queried on each $y \in U$ and $\mathbf{Pr}[R^{C \oplus \mathrm{QBF}}(x) = 1] \leqslant 1/4$ for every oracle $C \in Q(B, U)$. If such a U can be found, then reject; otherwise, accept.

To see why the described algorithm is correct, consider any set $U \subseteq \{0,1\}^n$ of size at most $8t^2$ such that $\mathbf{Pr}[R^{C \oplus \mathrm{QBF}}(x) = 1] \leqslant 1/4$ for every oracle $C \in Q(B, U)$. Suppose that x was not rejected in iteration l for any $l \leqslant 8t^2$. Then U must intersect each $U(l)$, $1 \leqslant l \leqslant 8t^2$, at some string y_l such that $B'(l)$ and B disagree on y_l. Otherwise, there would exist an extension B' of B_{i-1} that agrees with both $B'(l)$ on $U(l)$ and B on U, but this is impossible by Lemma 35. All such y_l's must be distinct, since $B'(l)$ agrees with B on every $z \in U(l) \cap U(m)$ for $m < l$. Thus, after $8t^2$ iterations, the value of B is known for every $y \in U$, and so x will be rejected. $\square$

The proof of Theorem 21 is now complete.