Electronic Colloquium on Computational Complexity, Report No. 36 (2002)

PP-lowness and a simple definition of AWPP

Stephen A. Fenner*
University of South Carolina

May 30, 2002

Abstract

We show that the counting classes **AWPP** and **APP** [Li93] are more robust than previously thought. Our results identify a sufficient condition for a language to be low for **PP**, and we show that this condition is at least as weak as other previously studied criteria. Our results imply that **AWPP** $\subseteq$ **APP**, and thus **APP** contains all other established subclasses of **PP**-low. We also show that **AWPP** and **APP** are Σ_2^0 definable classes. Our results are reminiscent of amplifying certainty in probabilistic computation.

Keywords: counting complexity, counting classes, PP, AWPP, PP-low

1 Introduction

Our main result is

Theorem 1.1 *A language L is low for* **PP** *if there are a polynomial p and a function $g \in \mathrm{Gap}\mathbf{P}$ [FFK94] such that*

$$x \in L \Rightarrow 2/3 \leq g(x)/2^p \leq 1,$$
$$x \notin L \Rightarrow 0 \leq g(x)/2^p \leq 1/3$$

for all $x \in \Sigma^*$, where $p = p(|x|)$.

The $\frac{1}{3}$–$\frac{2}{3}$ separation can be replaced with any constant positive separation, or even $\frac{1}{\mathrm{Poly}(|x|)}$. Also, 2^p can be replaced with any Gap**P** function which depends only on the length of x. Previously, the least known separation on $g(x)/2^p$ sufficient for **PP**-lowness is 2^{-r} to $1-2^{-r}$, where r is an arbitrary polynomial chosen before g and p (see Definition 1.2, below).

*Partially supported by South Carolina CHE SCRIG Grant R-01-0256 and by ARO DAAD 190210048.
Computer Science and Engineering Department, Columbia, SC 29208 USA. Email: fenner@cse.sc.edu.

ISSN 1433-8092

To our knowledge, ours is the weakest known sufficient criterion for **PP**-lowness involving constraints on a Gap**P** function. Our results build upon those of Li [Li93] and give simpler definitions of the counting classes **AWPP** and **APP** [Li93, FFKL93], whence we show that **AWPP** $\subseteq$ **APP**.

There are some interesting **NP** problems, Graph Isomorphism particularly, that are known to be low for **PP** [KST92], but it is unknown whether an **NP**-complete problem is **PP**-low. What is it about a language that makes it **PP**-low? A good approach to showing **PP**-lowness of a language L is to put L into a complexity class which is already known to contain only **PP**-low sets. To make it easy to do this, we want the largest such class(es) that we can find.

BPP consists entirely of **PP**-low sets, but so do various counting classes like **SPP**, or better yet **WPP** [FFK94]. Köbler *et al.* showed the **PP**-lowness of Graph Isomorphism by putting it into **WPP**.

The complexity classes **AWPP** (Definition 1.2 below) and **APP** were defined by Li [Li93, FFKL93], who was in search of big classes of **PP**-low sets. Li showed that **AWPP** and **APP** are subclasses of **PP**-low, and also contain **BPP** and all the other known subclasses of **PP**-low, including those mentioned above. (**AWPP** is really an analogue of the class **BPP** defined using Gap**P** functions instead of acceptance probabilities.) Later it was shown that there is an oracle G such that $\mathbf{P}^G = \mathbf{AWPP}^G$ but the polynomial hierarchy is infinite relative to G [FFKL93]. More recently, Fortnow and Rogers [FR99] showed that the class **BQP** of languages efficiently decidable by quantum computers with bounded error probability [BV97] is contained in **AWPP**. This means that all efficiently quantum computable languages are **PP**-low, and furthermore $\mathbf{P}^G = \mathbf{BQP}^G$ for the oracle G mentioned above.

Definition 1.2 (Li) *A language L is in* **AWPP** *if and only if, for every polynomial r there is a polynomial p and a* Gap**P** *function g such that, for all $x \in \Sigma^*$,*

$$\begin{aligned} x \in L &\Rightarrow 1 - 2^{-r} \leq g(x)/2^p \leq 1, \\ x \notin L &\Rightarrow 0 \leq g(x)/2^p \leq 2^{-r}, \end{aligned}$$

where $p = p(|x|)$ and $r = r(|x|)$.

The complexity of Definition 1.2 is irksome. For example, it is not even clear from the definition that **AWPP** is a Σ_2^0 definable class, whereas all the usual complexity classes are Σ_2^0. This definition appeared necessary, however, to obtain **PP**-lowness for **AWPP** languages. (Li gave other characterizations of **AWPP**, but they all involve universal quantification over the "error" polynomial r.) One would prefer to replace 2^{-r} and $1 - 2^{-r}$ above with constant fractions such as $\frac{1}{3}$ and $\frac{2}{3}$, giving a simpler Σ_2^0 definition of **AWPP** more closely analogous with **BPP**, but it was not known whether this could be done.

We show that one can indeed make such a replacement.

Theorem 1.3 *A language L is in* **AWPP** *if and only if there exist a polynomial p, and* Gap**P** *function g such that, for all $x \in \Sigma^*$,*

$$\begin{aligned} x \in L &\Rightarrow 2/3 \leq g(x)/2^p \leq 1, \\ x \notin L &\Rightarrow 0 \leq g(x)/2^p \leq 1/3, \end{aligned}$$

where $p = p(|x|)$.

Theorem 1.1 follows immediately from this and the **PP**-lowness results of Li [Li93]. We prove similar results for **APP** and as a corollary, we get that **AWPP** $\subseteq$ **APP**. Thus **APP** contains all other established complexity classes of **PP**-low sets.

To show Theorem 1.3, we iterate the polynomial $h(x) = 3x^2 - 2x^3$ to "squeeze" the Gap**P** function g toward 0 and toward 2^p, thus increasing the separation between acceptance and rejection. Iterating the polynomial h or a similar polynomial $4x^3 + 3x^4$ is a technique that has been used several times before to squeeze error in the context of modular arithmetic [Tod91, Yao90, For97]. Here we use it in the nonmodular setting.

2 Preliminaries

We let $\Sigma = \{0, 1\}$, and for $x \in \Sigma^*$ we write $|x|$ for the length of x. We may identify Σ^* with either $\mathbb{N}$ or with $\mathbb{Z}$ via standard binary encodings. We use standard complexity theoretic notation, and we assume knowledge of complexity classes, counting classes, and Gap**P** [FFK94]. In particular, we let **FP** be the class of all polynomial-time computable functions, and we fix a standard pairing function—a bijection $\langle\cdot,\cdot\rangle : \Sigma^* \times \Sigma^* \to \Sigma^*$ that is polynomial time computable and polynomial time invertible—which allows us to identify Σ^* with $\Sigma^* \times \Sigma^*$. We also fix some method of coding a finite sequence of strings $c_1, \ldots, c_n \in \Sigma^*$ as a single string $[c_1, \ldots, c_n] \in \Sigma^*$ so that $|[c_1, \ldots, c_n]| \in \mathcal{O}(n(1 + \max\{|c_i|\}))$. For any function f, define

$$f^{(n)} = \underbrace{f \circ \cdots \circ f}_{n},$$

for any integer $n \geq 0$ ($f^{(0)}$ is the identity function).

All logarithms are to base 2. All polynomials that we mention are in $\mathbb{Z}[x]$.

2.1 The Polynomial $3x^2 - 2x^3$

We briefly look at the properties of the polynomial $h(x) = 3x^2 - 2x^3$. The function h maps the interval $[0, 1]$ onto $[0, 1]$ in a monotone increasing way, and the graph of h on $[0, 1]$ is an S-shaped curve that is rotationally symmetric about the point $\left(\frac{1}{2}, \frac{1}{2}\right)$, that is, $h(1 - x) = 1 - h(x)$. The derivative of h vanishes at 0 and at 1. For any $0 < \epsilon < \frac{1}{2}$, define the *error set* $E_\epsilon = [0, \epsilon] \cup [1 - \epsilon, 1]$. Obviously, $0 < \epsilon_1 \leq \epsilon_2 < \frac{1}{2}$ implies $E_{\epsilon_1} \subseteq E_{\epsilon_2}$. It is also clear by symmetry that $h(E_\epsilon) = E_{h(\epsilon)} \subseteq E_\epsilon$. Let $\epsilon_i = h^{(i)}(\epsilon)$ for $i \geq 0$. Since $\epsilon_{i+1} < 3\epsilon_i^2$, we get by induction that $0 < \epsilon_i < \frac{1}{3}(3\epsilon)^{2^i}$ for all $i \geq 0$, and thus if $\epsilon \leq \frac{1}{6}$,

$$\epsilon_i \leq \frac{(3\epsilon)^{2^i}}{3} \leq \frac{2^{-2^i}}{3}.$$

If $\frac{1}{6} < \epsilon < \frac{1}{2}$, then $\epsilon_k \leq \frac{1}{6}$ for any integer $k \geq -4\left(1 + \log\left(\frac{1}{2} - \epsilon\right)\right)$. One way to see this is from the fact that if $\frac{1}{6} < x < \frac{1}{2}$, then $\frac{1}{2} - h(x) \geq \frac{5}{4}\left(\frac{1}{2} - x\right)$. We summarize these results in the following lemma:

Lemma 2.1 *For any positive $\delta < 1$, any $n \in \mathbb{N}$, and any integer $k \geq n + 4\log\frac{1}{\delta}$,*

$$0 < h^{(k)}\left(\frac{1-\delta}{2}\right) < 2^{-2^n}.$$

The coefficients of the polynomial $h^{(i)}$ are easy to compute in a way that we make precise in Section 2.2. This will imply that $h^{(i)}(f(x))$ is in Gap**P** whenever $f(x)$ is, where i is chosen appropriately depending on x.

2.2 Closure of GapP Under Iterated Polynomial Composition

Definition 2.2 *Let p be a polynomial. The* representation $\mathrm{rep}(p)$ *of p is a string in Σ^* defined as follows:*

$$\mathrm{rep}(p) = \begin{cases} [\,] & \text{if } p = 0, \\ [1^d, c_0, \ldots, c_d] & \text{if } p(x) = \sum_{j=0}^{d} c_j x^j \text{ with } c_d \neq 0. \end{cases}$$

Note that $|\mathrm{rep}(p)|$ bounds the degree of p.

The next few lemmas are crucial for our results. They are stated in more generality than we need here, as they may find use elsewhere.

Definition 2.3 *Let $p_0, p_1, p_2, \ldots$ be a sequence of polynomials. We say that $\{p_i\}_{i\in\mathbb{N}}$ is* ptime representable *if there is an **FP** function r such that $r(1^i) = \mathrm{rep}(p_i)$ for all $i \in \mathbb{N}$.*

Definition 2.4 *Let $p_0, p_1, p_2, \ldots$ be a sequence of polynomials. We say that $\{p_i\}_{i\in\mathbb{N}}$ is* Gap**P** representable *if there is a polynomial d and a* Gap**P** *function c such that, for all $i \in \mathbb{N}$,*

$$p_i(x) = \sum_{j=0}^{d(i)} c(1^i, 1^j) x^j.$$

The following lemma is obvious.

Lemma 2.5 *If $p_0, p_1, \ldots$ is ptime representable, then it is* Gap**P** *representable (indeed, via a function $c \in$ **FP**).*

Lemma 2.6 *If $p_0, p_1, p_2, \ldots$ is a* Gap**P** *representable family of polynomials and f is a* Gap**P** *function, then the function*

$$g(x) = p_{|x|}(f(x))$$

is also in Gap**P**.

Proof: This follows quickly from other known closure properties of Gap**P** [FFK94]. Since Gap**P** is closed under uniform polynomial size products, the function $e(x, 1^i) = \prod_{j=0}^{i-1} f(x) = f(x)^i$ is also in Gap**P** [FFK94, Corollary 3.8].

Let polynomial d and Gap**P** function c be as in Definition 2.4. Fix $x \in \Sigma^*$ of length n. Then

$$g(x) = p_n(f(x)) = \sum_{j=0}^{d(n)} c(1^n, 1^j) e(x, 1^j),$$

which is a uniform sum of products of Gap**P** functions. Hence, $g \in$ Gap**P**. □

Lemma 2.7 *Let p be any polynomial and let $s \in$* **FP** *be such that $s(x) \in \mathcal{O}(\log|x|)$. Then the sequence of polynomials $\{p^{(s(1^n))}\}_{n\in\mathbb{N}}$ is ptime representable.*

Proof: Fix $p(x) = \sum_{j=0}^{d} a_j x^j$ for constant $d > 0$ (the case for $d = 0$ is trivial) and $a_j \in \mathbb{Z}$ with $a_d \neq 0$ (the case for $p = 0$ is also trivial). Clearly, it is easy (polynomial time) to compute a representation for the composition $p \circ q$ of p with another polynomial q, given a representation for q. To compute a representation of $p^{(s(1^n))}$ on input 1^n, we start with a representation of the polynomial x, then repeatedly compose with p on the left $s(1^n)$ times. This can be all be done in time polynomial in n provided the intermediate representations do not get too large.

Suppose q is a polynomial of degree m. The composition $p \circ q$ then has degree md, and the largest absolute value of a coefficient in the composition can be seen to be bounded by $(d+1)a((m+1)b)^d$, where a and b are the largest absolute values of the coefficients of p and of q respectively. Recalling that d and a are constants, we get that $(d+1)a((m+1)b)^d \in \mathcal{O}\left(m^d b^d\right)$. It now follows by induction on $i \geq 0$ that $p^{(i)}$ has degree d^i, and all its coefficients have absolute value in $\mathcal{O}\left(C^{i^2 d^i}\right)$ for some constant C depending only on p. This immediately gives us an upper bound in $\mathcal{O}\left(i^2 d^{2i}\right)$ on the size of the representation of $p^{(i)}$. In the algorithm, $i \leq s(1^n) \in \mathcal{O}(\log n)$, so each representation in the algorithm has size in $\mathcal{O}\left((\log n)^2 d^{k \log n}\right)$ for some constant k. This is clearly polynomial in n, and so the algorithm runs in polynomial time. □

We will not iterate h itself but instead a scaled version of h, whence we need the following lemma:

Lemma 2.8 *Let $p_0, p_1, p_2, \ldots$ be a* Gap**P** *representable family of polynomials with degrees bounded by a polynomial d. Suppose s is a* Gap**P** *function outputting positive values. Then the family of polynomials $q_0, q_1, q_2, \ldots$ is* Gap**P** *representable, where*

$$q_i(x) = s_i^{d_i} p(x/s_i),$$

for all $i \in \mathbb{N}$, where $s_i = s(1^i)$ and $d_i = d(i)$.

Proof: Let $c \in \mathrm{Gap}\mathbf{P}$ such that $p_i(x) = \sum_{j=0}^{d_i} c(1^i, 1^j)x^j$. Then

$$q_i(x) = \sum_{j=0}^{d^i} c(1^i, 1^j) s_i^{d_i - j} x^j.$$

Setting $c'(1^i, 1^j) = c(1^i, 1^j)s_i^{d_i-j}$, it is clear by the closure properties of Gap**P** that $c' \in \mathrm{Gap}\mathbf{P}$ and c' and d witness that the family of q_i is Gap**P** representable. □

3 Main Results

3.1 AWPP

Theorem 1.3 immediately follows from the next theorem.

Theorem 3.1 *Let L be a language. $L \in$* **AWPP** *if and only if there are polynomials $u, q > 0$ and a* Gap**P** *function f such that, for all $x \in \Sigma^*$ with $n = |x|$,*

$$x \in L \;\Rightarrow\; \frac{1+\delta_n}{2} \le \frac{f(x)}{2^{q(n)}} \le 1,$$
$$x \notin L \;\Rightarrow\; 0 \le \frac{f(x)}{2^{q(n)}} \le \frac{1-\delta_n}{2},$$

where $\delta_n = 1/u(n)$.

Proof: We prove the "if" part; the "only if" part is trivial. Let L, u, q, and f be as in Theorem 3.1. We show that L satisfies Definition 1.2 for any polynomial r. We may assume that $r(n) > 0$ for all $n \in \mathbb{N}$. Let b be a polynomial such that $b(n)$ is an upper bound on $r(n)/\delta_n^4$ for all $n \in \mathbb{N}$ with $\delta_n = 1/u(n)$. For $n \in \mathbb{N}$, define

$$k_n = \lceil \log b(n) \rceil \ge \log r(n) + 4 \log \frac{1}{\delta_n} = \log(r(n)u(n)^4).$$

The family $h^{(k_0)}, h^{(k_1)}, h^{(k_2)}, \ldots$ is ptime representable by Lemma 2.7, and hence Gap**P** representable by Lemma 2.5.

Set $\epsilon_n = (1 - \delta_n)/2$. By Lemma 2.1 we have $h^{(k_n)}(\epsilon_n) < 2^{-r(n)}$.

Noting that $h^{(k_n)}$ has degree $3^{k_n} \leq 3b(n)^2$, we let z_n be the polynomials

$$z_n(y) = 2^{3q(n)b(n)^2} h^{(k_n)} \left(\frac{y}{2^{q(n)}}\right).$$

By Lemma 2.8, $z_0, z_1, z_2, \ldots$ is Gap**P** representable.

Now for all $n \in \mathbb{N}$ and $x \in \Sigma^*$ of length n, we define

$$\begin{aligned} p(n) &= 3q(n)b(n)^2, \\ g(x) &= z_n(f(x)). \end{aligned}$$

It follows from Lemma 2.6 that $g \in \text{Gap}\mathbf{P}$. Finally,

$$\begin{aligned} x \notin L &\Rightarrow 0 \leq f(x)/2^{q(n)} \leq \epsilon_n \\ &\Rightarrow 0 \leq h^{(k_n)}(f(x)/2^{q(n)}) \leq 2^{-r(n)} \\ &\Rightarrow 0 \leq g(x)/2^{p(n)} \leq 2^{-r(n)}, \end{aligned}$$

and similarly, $x \in L \Rightarrow 1 - 2^{-r(n)} \leq g(x)/2^{p(n)} \leq 1$. Therefore $L \in \mathbf{AWPP}$. □

Corollary 3.2 **AWPP** *is a* Σ_2^0 *definable class.*

3.2 APP

Definition 3.3 (Li [Li93]) *The class* **APP** *consists of all languages* L *such that for all polynomials* r *there exist* $f, g \in \text{Gap}\mathbf{P}$ *such that* $g(1^n) > 0$ *for all* $n \in \mathbb{N}$, *and for all* n, x *with* $n \geq |x|$,

$$\begin{aligned} x \in L &\Rightarrow 1 - 2^{-r(n)} \leq \frac{f(x, 1^n)}{g(1^n)} \leq 1, \\ x \notin L &\Rightarrow 0 \leq \frac{f(x, 1^n)}{g(1^n)} \leq 2^{-r(n)}. \end{aligned}$$

Li showed that all **APP** languages are **PP**-low [Li93]. **APP** is similar to **AWPP** but handles the error threshold with an extra parameter. We show that both the polynomial r and this extra parameter can be dispensed with. As a corollary, we get that **AWPP** $\subseteq$ **APP**.

Theorem 3.4 *Let* L *be a language. The following are equivalent:*

1. $L \in \mathbf{APP}$.

2. *There exist* $f, g \in \text{Gap}\mathbf{P}$ *and a polynomial* $u > 0$ *such that for all* $x \in \Sigma^*$ *and* $n \in \mathbb{N}$ *with* $n \geq |x|$, $g(1^n) > 0$ *and*

$$\begin{aligned} x \in L &\Rightarrow \frac{1 + \delta_n}{2} \leq \frac{f(x, 1^n)}{g(1^n)} \leq 1, \\ x \notin L &\Rightarrow 0 \leq \frac{f(x, 1^n)}{g(1^n)} \leq \frac{1 - \delta_n}{2}, \end{aligned}$$

   where $\delta_n = 1/u(n)$.

3. *There exist $f, g \in \mathrm{Gap}\mathbf{P}$ and a polynomial $u > 0$ such that for all $x \in \Sigma^*$, $g(1^{|x|}) > 0$ and*

$$x \in L \Rightarrow \frac{1+\delta_{|x|}}{2} \leq \frac{f(x)}{g(1^{|x|})} \leq 1,$$
$$x \notin L \Rightarrow 0 \leq \frac{f(x)}{g(1^{|x|})} \leq \frac{1-\delta_{|x|}}{2},$$

where $\delta_{|x|} = 1/u(|x|)$.

Proof: (2) $\Rightarrow$ (1): Let $f, g \in \mathrm{Gap}\mathbf{P}$ and u be as in (2). Let $r > 0$ be a fixed polynomial. Define b and $k_0, k_1, k_2, \ldots$ as in the proof of Theorem 3.1. Let $z_0, z_1, z_2, \ldots$ be the family of polynomials

$$z_n(y) = g(1^n)^{3b(n)^2} h^{(k_n)} \left(\frac{y}{g(1^n)} \right),$$

which is $\mathrm{Gap}\mathbf{P}$ representable by Lemma 2.8 as before. Now for $x \in \Sigma^*$ and $n \in \mathbb{N}$ with $n \geq |x|$ let

$$\begin{aligned} g'(1^n) &= g(1^n)^{3b(n)^2} \\ f'(x, 1^n) &= z_n(f(x, 1^n)). \end{aligned}$$

Both g' and f' are in $\mathrm{Gap}\mathbf{P}$, the latter inclusion following from Lemma 2.6. Then we have, as in the proof of Theorem 3.1,

$$\begin{aligned} x \notin L &\Rightarrow 0 \leq f(x, 1^n)/g(1^n) \leq (1 - \delta_n)/2 \\ &\Rightarrow 0 \leq h^{(k_n)}(f(x, 1^n)/g(1^n)) \leq 2^{-r(n)} \\ &\Rightarrow 0 \leq f'(x, 1^n)/g'(1^n) \leq 2^{-r(n)}, \end{aligned}$$

and similarly, $x \in L \Rightarrow 1 - 2^{-r(n)} \leq f'(x, 1^n)/g'(1^n) \leq 1$. Thus $L \in \mathbf{APP}$ witnessed by f' and g'.

(3) $\Rightarrow$ (2): Let $f, g \in \mathrm{Gap}\mathbf{P}$ and u be as in (3). For $x \in \Sigma^*$ and $n \geq |x|$ define

$$\begin{aligned} g'(1^n) &= \prod_{i=0}^{n} g(1^i), \\ f'(x, 1^n) &= f(x) g'(1^n)/g(1^{|x|}). \end{aligned}$$

Clearly, $f', g' \in \mathrm{Gap}\mathbf{P}$, and together with u witness that L satisfies (2).

(1) $\Rightarrow$ (3): Let f and g be as in (1) when $r(n)$ is the constant 2. Define

$$\begin{aligned} u &= 2, \\ f'(x) &= f(x, 1^{|x|}). \end{aligned}$$

Then f', g, and u witness that L satisfies (3). □

Corollary 3.5 **APP** *is a* Σ_2^0 *definable class.*

Corollary 3.6 **AWPP** $\subseteq$ **APP**.

Proof: Compare Theorem 3.1 with item (3) in Theorem 3.4, setting $g(1^n) = 2^{q(n)}$. □

4 Conclusions and Open Questions

We have seen that both classes **AWPP** and **APP** can be defined much more simply and naturally than they were originally. This added robustness in the definitions makes both classes much more interesting. Li showed that the denominator $2^{q(|x|)}$ in the definition of **AWPP** can be replaced with an arbitrary positive **FP** function of x [Li93]. Combining with the current results, we see that the only difference between **AWPP** and **APP** is that in the latter, the denominator can be any Gap**P** function of $1^{|x|}$. (Li also showed that if we allow the denominator to be any Gap**P** function of x, then we get the class **PP** [Li93].)

Since they solve the issue of error amplification in general, our results make it technically much easier to prove membership in **AWPP** or **APP**, and hence lowness for **PP**. For example, the proof that **BQP** $\subseteq$ **AWPP** of Fortnow and Rogers [FR99] can be simplified by ignoring the error amplification properties of **BQP**. We are not, however, aware of any specific concrete problem that is now known to be low for **PP** as a direct consequence of our results, and we would be very interested in finding such a problem.

Are **AWPP** and **APP** equal? Our results boil this question down to the following: "Can a Gap**P** function that only depends on $|x|$ be replaced by an **FP** function in the denominator in item (3) of Theorem 3.4?". Such a result would certainly add to the robustness of **AWPP**.

Finally, we know of no concrete problem in **AWPP** or in **APP** that is not also known to be in a previously studied subclass. Discovering such a problem would increase the importance of these classes significantly.

References

[BV97] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM J. Comp.*, 26(5):1411–1473, 1997.

[FFK94] S. Fenner, L. Fortnow, and S. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1):116–148, 1994. An earlier version appeared in *Proceedings of the 6th Annual IEEE Structure in Complexity Theory Conference*, 1991, pp. 30–42.

[FFKL93] S. Fenner, L. Fortnow, S. Kurtz, and L. Li. An oracle builder's toolkit. In *Proceedings of the 8th IEEE Structure in Complexity Theory Conference*, pages 120–131, 1993.

[For97] L. Fortnow. Counting complexity. In L. A. Hemaspaandra and A. L. Selman, editors, *Complexity Theory Retrospective II*. Springer-Verlag, 1997.

[FR99] L. Fortnow and J. Rogers. Complexity limitations on quantum computation. *Journal of Computer and System Sciences*, 59(2):240–252, 1999.

[KST92] J. Köbler, U. Schöning, and J. Torán. Graph Isomorphism is low for PP. *Computational Complexity*, 2(4):301–330, 1992.

[Li93] L. Li. On the counting functions. Technical Report TR-93-12, The University of Chicago, 1993. PhD thesis, available at http://www.cs.uchicago.edu/research/publications/techreports/TR-93-12.

[Tod91] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[Yao90] A. Yao. On ACC and threshold circuits. In *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science*, pages 619–631, New York, 1990. IEEE.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'