PP-lowness and a simple definition of AWPP

Stephen A. Fenner*
University of South Carolina

June 16, 2003

Abstract

We show that the counting classes **AWPP** and **APP** [FFKL93, 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 also imply that **AWPP** $\subseteq$ **APP**, and thus **APP** contains many other established subclasses of **PP**-low, thereby reducing several different lowness results to membership in **APP**. We give a simplified proof of an earlier result of Li, which extends a result of Köbler *et al.*: all sparse co-$\mathrm{C}_{=}$**P** languages are in **APP**, and are thus **PP**-low. We also show that **AWPP** and **APP** are Σ_2^0-definable classes. Some of our results are reminiscent of amplifying certainty in probabilistic computation.

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

1 Introduction

A language L is *low* for a relativizable complexity class $\mathcal{C}$ (L is $\mathcal{C}$-*low*) if $\mathcal{C}^L = \mathcal{C}$. That is, L provides no help as an oracle for a $\mathcal{C}$-computation. Our interest here is in lowness for the counting class **PP** [Gil77].

There are some important **NP** problems—Graph Isomorphism particularly—that are known to be low for **PP** [KST92]. It is also known that all sparse **NP** problems are low for **PP** [KSTT92], 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 [KSTT92], 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** [KST92].

*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.

One of our main results closely relates **PP**-lowness with the behavior of functions in the class Gap**P**, a variant of #**P** [FFK94] (see Definition 2.1). We show

Theorem 1.1 *A language L is low for* **PP** *if there are a polynomial p and a function $g \in$ Gap**P** such that*

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

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 **FP** function, or even with any Gap**P** function which depends only on the length of x. Previously, the least separation on $g(x)/2^p$ known to be sufficient for **PP**-lowness was 2^{-r} to $1 - 2^{-r}$, where r is an arbitrary polynomial chosen before g and p [Li93] (see Definition 1.2, below). 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] (see Definitions 1.2 and 1.3, below), whence we show that **AWPP** $\subseteq$ **APP**.

The class **AWPP** is significant in relation to the quantum computing model. It holds the distinction of being the smallest known "standard" (i.e., nonquantum) complexity class which contains the class **BQP** of all decision problems efficiently solvable by quantum computers with bounded error probability [BV97, FR99]. **AWPP** is an analogue of the class **BPP** defined using Gap**P** functions instead of acceptance probabilities. **AWPP** and the related complexity class **APP** were originally defined by Li [FFKL93, Li93], who was in search of big classes of **PP**-low sets. Li showed that **AWPP** and **APP** are subclasses of **PP**-low (see [Fen02a] for a published proof of the **PP**-lowness of **AWPP**) and also contain **BPP** and other known subclasses of **PP**-low, including those mentioned earlier [Li93]. It was later 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 the containment **BQP** $\subseteq$ **AWPP** mentioned above. This means that all efficiently quantum computable languages are **PP**-low, and furthermore $\mathbf{P}^G = \mathbf{BQP}^G$ for the oracle G just described.

The analogies with **BPP** are evident from the following definitions of **AWPP** and **APP**:

Definition 1.2 (Li [FFKL93]) *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 $r = r(|x|)$.

Definition 1.3 (Li [Li93]) *A language L is in* **APP** *if and only if, for all polynomials r there exist $f, g \in$ Gap**P** such that, for all n and x with $n \geq |x|$, $g(1^n) > 0$ and*

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

Definitions 1.2 and 1.3 are awkward, and they are overly complex in a very precise sense: both appear to require three alternating first-order quantifiers ($\forall r \exists p, g \forall x \ldots$ and $\forall r \exists f, g \forall n, x \ldots$, respectively) before an L-computable predicate. In the language of descriptive set theory, we can then only say that **AWPP** and **APP** are "Π_3^0-definable." All the usual complexity classes are Σ_2^0*-definable*, that is, definable using only two alternating first-order quantifiers ($\exists\forall$) in front of an L-computable predicate. Definitions 1.2 and 1.3 seemed necessary, however, to obtain **PP**-lowness for languages in these classes. (Li gave other characterizations of **AWPP**, but they all involve a third (universal) quantification over the "error" polynomial r.) One would prefer to replace 2^{-r} and $1-2^{-r}$ in Definition 1.2 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 1.4 *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 simplifying results for **APP**, and as a corollary we get that **AWPP** $\subseteq$ **APP** (Corollary 3.6).

We will prove a stronger statement, Theorem 3.1, which immediately implies Theorem 1.4. It was shown in [FFKL93, Li93] that Definition 1.2 does not change if we replace 2^p in the denominator with any positive **FP** function of x. We will show by a shorter proof (Theorem 3.3) that the same replacement can be made in Theorem 1.4 as well.

Köbler *et al.* showed that all sparse **NP** sets are **PP**-low [KSTT92]. Li [Li93] showed that all sparse co-C$_=$**P** languages are in **APP** and hence **PP**-low. This strengthens [KSTT92]; it is well-known that **NP** $\subseteq$ co-C$_=$**P** (the latter is sometimes denoted C$_{\neq}$**P**), but the two classes are probably not equal. We use our results to provide a shorter and simpler proof of Li's result.[1] Thus **APP** contains all these other subclasses of **PP**-low: **FewP** $\subseteq$ **SPP** $\subseteq$

[1] The journal version of this paper [Fen02b] was published before Li's result was made known to the author.

WPP $\subseteq$ **AWPP**, **BPP** $\subseteq$ **AWPP**, **NP** $\cap$ **SPARSE** $\subseteq$ co-C$_=$**P** $\cap$ **SPARSE** [FFK94, FFKL93, KSTT92], where **SPARSE** is the class of all sparse languages (see below).

To show Theorem 1.4, 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. The polynomial h is perhaps the simplest example of an "amplifying polynomial." The technique of iterating polynomials such as h or a similar polynomial $4x^3 + 3x^4$ has been used several times before to squeeze error in the context of modular arithmetic [Tod91, Yao90, For97]. Polynomials similar to h have also been used in the nonmodular setting to amplify probability. For example, they were used to show that small monotone circuits exist for the majority function [Val84]. Here, we use h in the nonmodular setting to "amplify" Gap**P** functions rather than probabilities.

2 Preliminaries

We let $\Sigma = \{0, 1\}$, whence Σ^* is the set of all binary strings. For $x \in \Sigma^*$ we write $|x|$ for the length of x, and for integers $n \geq 0$ we let Σ^n denote the set of all strings of length n. We may identify Σ^* with either $\mathbb{N}$ (the natural numbers, including zero) or with $\mathbb{Z}$ (the integers) via standard binary encodings. We use standard complexity theoretic notation, and we assume knowledge of standard complexity classes, especially counting classes such as #**P**, **PP** [Gil77], and C$_=$**P** [Wag86]. We will use the function class Gap**P**, which is the closure of #**P** under subtraction.

Definition 2.1 ([FFK94]) *A function $f : \Sigma^* \to \mathbb{Z}$ is in* Gap**P** *if and only if there are $g, h \in$ #**P** such that $f(x) = g(x) - h(x)$ for all $x \in \Sigma^*$.*

Equivalently, f is in Gap**P** *if there is a nondeterministic polynomial-time Turing machine M such that, for all $x \in \Sigma^*$, $f(x)$ is the number of accepting paths minus the number of rejecting paths (i.e., the "gap") of M on input x.*

Gap**P** is closed under negation, uniform exponential size sums, and uniform polynomial sized products. See [FFK94] for more information about Gap**P**, its precise closure properties, and its relationship to the counting classes mentioned above. 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 $|[c_1, \ldots, c_n]| \in \mathcal{O}\left(n\left(1 + \max\{|c_i|\}\right)\right)$. 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).

We let $\|A\|$ denote the cardinality of a finite set A. A language $L \subseteq \Sigma^*$ is *sparse* if there is a polynomial p such that $\|L \cap \Sigma^n\| \leq p(n)$ for all $n \geq 0$. We let **SPARSE** denote the class of all sparse languages.

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 as follows. Let $a(y) = \frac{1}{2} - y$ and let $j(y) = (a \circ h \circ a^{(-1)})(y) = \frac{1}{2} - h\left(\frac{1}{2} - y\right)$. For $0 < y < \frac{1}{2}$, we have $y < j(y) < \frac{1}{2}$, and if $0 < y < \frac{1}{3}$, we have $j(y) > \frac{23}{18}y > \frac{5}{4}y$. Thus for fixed $0 < y < \frac{1}{3}$ and $k > 0$, we have $\frac{1}{2} - h^{(k)}\left(\frac{1}{2} - y\right) = j^{(k)}(y) > \left(\frac{5}{4}\right)^k y$, provided $j^{k-1}(y) < \frac{1}{3}$. It follows that $\frac{1}{3} \leq j^{(k)}(y) < \frac{1}{2}$ if

$$k \geq \frac{\log\frac{1}{3} - \log y}{\log\frac{5}{4}},$$

which is easily checked to be true for $k \geq -4(1 + \log y)$.

We summarize these results in the following lemma:

Lemma 2.2 *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 $h^{(i)}$ can be applied to a suitably scaled Gap**P** function to yield another suitably scaled Gap**P** function, where i is chosen appropriately depending on x.

2.2 Closure of Gap**P** Under Iterated Polynomial Composition

Definition 2.3 *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 2.4 *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 2.5 *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 2.6 *If $p_0, p_1, \ldots$ is ptime representable, then it is* Gap**P** *representable (indeed, via a function $c \in$* **FP***).*

Lemma 2.7 *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 2.5. 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 2.8 *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 2.9 *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*

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

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

Proof: Let $c \in$ Gap**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 $c' \in$ Gap**P** and c' and d witness that the family of q_i is Gap**P** representable. □

3 Main Results

3.1 AWPP

Theorem 1.4 immediately follows from the next theorem.

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

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

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

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

To prove Theorem 3.1, we simply apply the polynomial function $h(y) = 3y^2 - 2y^3$ logarithmically many times to the value $\frac{f(x)}{2^{q(n)}}$ above. The amplifying properties of h serve to broaden the avoided interval $\left(\frac{1-\delta_n}{2}, \frac{1+\delta_n}{2}\right)$ so that it covers all but an exponentially small amount of the interval $[0, 1]$.

Proof: We prove the "if" part; the "only if" part is trivial. Let L, q, u, 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

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

The family $h^{(k_0)}, h^{(k_1)}, h^{(k_2)}, \ldots$ is ptime representable by Lemma 2.8, and hence Gap**P** representable by Lemma 2.6.

Set $\epsilon_n = (1 - \delta_n)/2$. By Lemma 2.2 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 2.9, $z_0, z_1, z_2, \ldots$ 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 2.7 that $g \in$ Gap**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 $L \in$ **AWPP**. □

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

We can strengthen Theorem 3.1 by allowing any **FP** function of x in the denominator, analogous with results in [FFKL93, Li93]. Theorem 3.3 provides the weakest sufficient condition (to our knowledge) for membership in **AWPP**. The proof is made easier by Theorem 3.1 itself.

Theorem 3.3 *Let L be a language. $L \in$ **AWPP** if and only if there is a polynomial $u > 0$, a* Gap**P** *function f, and a $d \in$ **FP** such that, for all $n \in \mathbb{N}$ and $x \in \Sigma^n$, $d(x) > 0$ and*

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

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

Proof: The "only if" part follows immediately from Theorem 3.1. Conversely, suppose u, f, and d satisfy the conclusion. Let q be a polynomial such that $d(x) \leq 2^{q(|x|)-1}$ for all $x \in \Sigma^*$. For all $x \in \Sigma^*$ with $n = |x|$, define

$$f'(x) = 2^{q(n)-1} + \left\lfloor \frac{2^{q(n)-1}}{d(x)} \right\rfloor (2f(x) - d(x)).$$

Then $f' \in$ Gap**P**, and

$$\frac{f'(x)}{2^{q(n)}} = \frac{1}{2} + \frac{d(x)}{2^{q(n)-1}} \left\lfloor \frac{2^{q(n)-1}}{d(x)} \right\rfloor \left(\frac{f(x)}{d(x)} - \frac{1}{2} \right). \tag{1}$$

Using (1) and the fact that $\frac{1}{2} < \frac{d(x)}{2^{q(n)-1}} \left\lfloor \frac{2^{q(n)-1}}{d(x)} \right\rfloor \leq 1$, a routine calculation shows that the conclusion of Theorem 3.1 is satisfied by q, f', and $\delta'_n = \frac{1}{2u(n)}$. Therefore, $L \in$ **AWPP**. □

3.2 APP

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. *$L \in$ **APP**.*
2. *There exist $f, g \in$ Gap**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*

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

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 Gap$\mathbf{P}$ representable by Lemma 2.9 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 Gap$\mathbf{P}$, the latter inclusion following from Lemma 2.7. 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). $\square$

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 $g(1^n) = 2^{q(n)}$. □

For both classes **AWPP** and **APP**, there is no reason why the gap in the allowed values of $f(x)/g(1^{|x|})$ need be centered at 1/2. For example, we have the following corollary to Theorem 3.4:

Corollary 3.7 *A language* L *is in* **APP** *if and only if there exist* $f, g \in$ Gap**P** *and constants* $0 \leq \lambda < \upsilon \leq 1$ *such that for all* $x \in \Sigma^*$, $g(1^{|x|}) > 0$ *and*

$$\begin{aligned} x \in L &\Rightarrow \upsilon \leq \frac{f(x)}{g(1^{|x|})} \leq 1, \\ x \notin L &\Rightarrow 0 \leq \frac{f(x)}{g(1^{|x|})} \leq \lambda. \end{aligned}$$

Proof: If $L \in$ **APP**, then by letting $r(n) = 2$ in Definition 1.3, we get $f', g \in$ Gap**P** such that for all $x \in \Sigma^*$,

$$\begin{aligned} x \in L &\Rightarrow \frac{3}{4} \leq \frac{f'(x, 1^{|x|})}{g(1^{|x|})} \leq 1, \\ x \notin L &\Rightarrow 0 \leq \frac{f'(x, 1^{|x|})}{g(1^{|x|})} \leq \frac{1}{4}. \end{aligned}$$

Setting $f(x) = f'(x, 1^{|x|})$, $\lambda = \frac{1}{4}$, and $\upsilon = \frac{3}{4}$ yields the conclusion of the corollary.

Conversely, suppose the conclusion of the corollary is satisfied for some $f', g', \lambda, \upsilon$. We "linearly adjust" the quantity $f'(x)/g'(1^{|x|})$. Without loss of generality, we may assume that $\lambda = a2^{-e}$ and $\upsilon = b2^{-e}$ for some integers a, b, e with $0 \leq a < b \leq 2^e$. If we set

$$\begin{aligned} f(x) &= 2^{e+1} f'(x) + (2^{e+1} - a - b) g'(x), \\ g(x) &= 2^{e+2} g'(x), \end{aligned}$$

then a straightforward calculation shows that Part (3) of Theorem 3.4 is satisfied with f, g, and any constant $\delta = \delta_{|x|} \leq (b-a)2^{-e-1}$. Thus $L \in$ **APP**. □

We conclude this section by using our results to give a shorter proof of the result by Li [Li93] that every sparse co-C$_=$**P** language is in **APP**, and hence is **PP**-low. We note here that Ogiwara [Ogi92] attempted to prove lowness of sparse sets in counting complexity classes including co-C$_=$**P**, but made little progress toward the question.

Theorem 3.8 (Li) co-C$_=$**P** $\cap$ **SPARSE** $\subseteq$ **APP**.

Proof: We adapt the basic technique in [KSTT92] (our proof was found independently of Li's proof). Let L be a sparse co-$\mathrm{C}_{=}\mathbf{P}$ language. It is known that there is an $h \in \mathrm{Gap}\mathbf{P}$ such that, for all $x \in \Sigma^*$, $x \in L$ iff $h(x) \neq 0$ [FFK94]. By squaring h, we may assume that $h(x) \geq 0$ for all x. Let p be a polynomial such that $\|L \cap \Sigma^n\| \leq p(n)$ for all $n \geq 0$. For all $n \geq 0$ and all x of length n, let

$$g(x) = \sum_{S \subseteq \Sigma^n \ \& \ \|S\| \leq p(n)} \ \prod_{y \in S} h(y), \tag{2}$$

and let

$$f(x) = \sum_{S \subseteq \Sigma^n \ \& \ \|S\| \leq p(n) \ \& \ x \notin S} \ \prod_{y \in S} h(y). \tag{3}$$

Since Gap$\mathbf{P}$ is closed under uniform exponential size sums and polynomial size products [FFK94], it is evident that $f, g \in \mathrm{Gap}\mathbf{P}$. Indeed, we could describe f and g equivalently in terms of machines. To compute g, for example, the machine first guesses a subset $S \subseteq \Sigma^n$, then generates a gap of $\prod_{y \in S} h(y)$ by simulating the machine for h sequentially for each element of S as input.

Note that (i) the only nonzero terms appearing on the right-hand sides of (2) or (3) are for $S \subseteq L$, (ii) $g(x) > 0$ since for $S = \emptyset$ the empty product contributes 1 to the sum, and (iii) $g(x)$ depends only on the length of x.

If $x \notin L$, then $h(x) = 0$, and so any term on the right-hand side of (2) where $x \in S$ is zero. Thus in this case it is clear that $f(x) = g(x)$. If $x \in L$, then clearly

$$\begin{aligned}
g(x) &= f(x) + \sum_{S \subseteq \Sigma^n \ \& \ \|S\| \leq p(n) \ \& \ x \in S} \ \prod_{y \in S} h(y) && (4)\\
&= f(x) + h(x) \left(\sum_{S \subseteq \Sigma^n \ \& \ \|S\| \leq p(n)-1 \ \& \ x \notin S} \ \prod_{y \in S} h(y) \right) && (5)\\
&= f(x) + h(x) \left(\sum_{S \subseteq \Sigma^n \ \& \ \|S\| \leq p(n) \ \& \ x \notin S} \ \prod_{y \in S} h(y) \right) && (6)\\
&= f(x) + h(x) f(x) && (7)\\
&\geq 2f(x). && (8)
\end{aligned}$$

The transition from (5) to (6) holds because there are at most $p(n) - 1$ strings in L of length n besides x, and so the terms appearing in (6) for $\|S\| = p(n)$ are all zero.

Putting the two cases together, we have, for all $x \in \Sigma^*$,

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

because $g(x) = g(1^{|x|})$. Thus $\overline{L} \in \mathbf{APP}$ by Corollary 3.7. **APP** is clearly closed under complements, so $L \in \mathbf{APP}$. □

Corollary 3.9 (Li) *All sparse languages in* co-C$_=$**P** *are low for* **PP**.

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**. For another example, the proof of Theorem 3.8 above would be more complicated had we just used the original definition of **APP**. We are not, however, aware of any interesting 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.

Acknowledgments

I would like to thank Frederic Green, Steven Homer, and Lance Fortnow for helpful and interesting discussions of this and related topics. Thanks also to Michael Saks and Leslie Valiant for bringing the techniques in [Val84] to my attention, and to an anonymous referee for posing the question of whether all sparse **NP** sets are in **APP**.

References

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

[Fen02a] S. A. Fenner. Counting complexity and quantum computation. In R. K. Brylinski and G. Chen, editors, *Mathematics of Quantum Computation*, chapter 8, pages 171–219. CRC Press, 2002.

[Fen02b] S. A. Fenner. PP-lowness and a simple definition of AWPP. *Theory of Computing Systems*, 2002. To appear. Also available as ECCC Report TR02-036.

[FFK94] S. Fenner, L. Fortnow, and S. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1):116–148, 1994.

[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. Accepted to *Information and Computation*. Draft available at http://www.cse.sc.edu/~fenner/papers/toolkit.ps.

[For97] L. Fortnow. Counting complexity. In L. A. Hemaspaandra and A. L. Selman, editors, *Complexity Theory Retrospective II*, pages 81–107. 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, cs.CC/9811023.

[Gil77] J. Gill. Computational complexity of probabilistic complexity classes. *SIAM Journal on Computing*, 6:675–695, 1977.

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

[KSTT92] J. Köbler, U. Schöning, S. Toda, and J. Torán. Turing machines with few accepting computations and low sets for PP. *Journal of Computer and System Sciences*, 44(2):272–286, 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.

[Ogi92] Mitsunori Ogiwara. *Studies of Counting Complexity Classes via Sets with Small Density*. PhD thesis, Tokyo Institute of Technology, 1992.

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

[Val84] L. G. Valiant. Short monotone formulæ for the majority function. *Journal of Algorithms*, 5:363–366, 1984.

[Wag86] K. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23:325–356, 1986.

[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.