1

Counting Complexity

Lance Fortnow[1]

ABSTRACT Traditionally, computational complexity theorists have looked at NP problems like three-coloring to determine whether some solution exists. In counting complexity, we ask how many solutions exist. This paper surveys the vast area of counting complexity, concentrating on a few of the most important results in the area.

1 Introduction

In 1979, Valiant [Val79a] defined the complexity class #P as a function class computing the number of accepting paths of a nondeterministic Turing machine. Valiant used this class to capture the complexity of the permanent function.

Counting complexity has since played an important role in computational complexity theory and theoretical computer science. The techniques used in counting complexity have significant applications in circuit complexity and the series of recent results on interactive proof systems.

In the first *Complexity Theory Retrospective*, Schöning [Sch90] gave us a taste of the "power of counting." Schöning's survey looked at counting as a technique in complexity theory. We instead will look at counting as a goal and study the techniques (mostly algebraic) that help us understand the computational complexity of counting.

In this short survey, we cannot hope to cover adequately the wealth of research in counting complexity. We focus on describing the basics: the two important counting function classes #P and GapP, and several important language classes defined in terms of these functions. We will describe the basic relationships and closure properties among these classes.

We focus in detail on perhaps the two most important results in counting complexity: Toda's [Tod91] result showing that one can reduce any language in the polynomial-time hierarchy to a #P function and Beigel, Reingold, and Spielman's [BRS95] theorem that PP is closed under union.

[1]Department of Computer Science, University of Chicago, 1100 E. 58th St., Chicago, IL 60637. Email: fortnow@cs.uchicago.edu. Supported in part by NSF grant CCR 92-53582.

We prove Toda's Theorem in three stages. First, we give a new algebraic proof of an important lemma by Valiant and Vazirani [VV86] that "randomly" reduces a #P function to its sign. We apply this lemma during the proof of a result of Toda and Ogihara [TO92] that shows that the polynomial-time hierarchy is in some sense probabilistically low for GapP functions. We then use this result with a variant of Toda's original proof to show how to reduce the polynomial-time hierarchy deterministically to a single GapP question.

This survey should give the reader a taste of the power of counting functions. We hope that the reader will take the concepts herein and learn more about counting complexity and apply the techniques not only to counting complexity but also to other areas in computational complexity theory and theoretical computer science.

2 Preliminaries

We assume that the reader has familiarity with the basic notions of Turing machines, nondeterminism, and basic complexity classes such as P and NP. The reader can find these notions in a basic undergraduate textbook such as that of Hopcroft and Ullman [HU79].

Fix $\Sigma = \{0,1\}$. We consider strings over Σ both as concatenations of characters and as representations of integers. We define the one-to-one correspondence as follows:

1. The string ϵ represents zero.
2. The string $0x$ represents the positive integer whose binary description is $1x$.
3. The string $1x$ represents the negative integer whose absolute value has binary description $1x$.

Note that for a number y the length of the representation of y, $|y|$, is roughly the logarithm of the absolute value of y.

We order strings lexicographically, i.e., $x < y$ if $|x| < |y|$ or $|x| = |y|$ and x precedes y in dictionary order. Note that we may have $x < y$ as strings but $x > y$ as integers.

Let $\langle x_1, \ldots, x_k \rangle$ for $k > 1$ be a standard tuple function, i.e., one-to-one, easily computable and invertible in all arguments. When we write a function as $f(x_1, \ldots, x_k)$, this should be interpreted as $f(\langle x_1, \ldots, x_k \rangle)$.

Let M be a nondeterministic Turing machine that runs in time bounded by some polynomial n^j for inputs of length n. Since most of the definitions in this paper are based on this type of machine, we give such machines the name NP machines. However, we will usually be interested more in the

number of accepting and possibly rejecting paths of M rather than whether the machine "accepts."

Let $\overline{M}$ represent the NP machine that simulates M but reverses M's decision to accept and reject. In other words, the accepting paths of $\overline{M}(x)$ are the rejecting paths of $M(x)$ and vice versa.

Let $\#M(x)$ be the number of accepting paths of M. We then have that $\#\overline{M}(x)$ is the number of rejecting paths. We also define the "Gap" of M by $\Delta M(x) = \#M(x) - \#\overline{M}(x)$.

Note that $\#M(x)$, $\overline{M}(x)$, and $\Delta M(x)$ all are bounded by $2^{|x|^j}$. 2 poly(n)

Let FP represent the class of polynomial-time computable functions.

We will also consider oracle Turing machines. An oracle Turing machine M using oracle A may write a string y on a special "oracle" tape and then enter a special "query" state. M will then enter a "yes" state if y is in A and a "no" state if y is not in A.

We define the relativized class NP^A as the set of languages that are accepted by polynomial-time nondeterministic Turing machines with access to oracle A. For a class $\mathcal{C}$, we define $\mathrm{NP}^{\mathcal{C}}$ as the union of all NP^A for A in $\mathcal{C}$. We use the notation $\mathrm{NP}^{A[k]}$ to mean that the NP machine can make only at most k queries to the oracle A on any computation path. We can define relativized versions of other complexity classes in a similar manner. When we relativize PSPACE computation, we only allow the PSPACE machine to write questions on the oracle tape whose lengths are bounded by a polynomial.

We can use oracle Turing machines to define other complexity classes such as the polynomial-time hierarchy [MS72]. We define Σ_i^p, Π_i^p, and Δ_i^p inductively as follows:

1. $\Sigma_0^p = \Pi_0^p = \Delta_0^p = \mathrm{P}$.
2. $\Sigma_{i+1}^p = \mathrm{NP}^{\Sigma_i^p}$.
3. $\Pi_{i+1}^p = \text{co-}\Sigma_{i+1}^p$.
4. $\Delta_{i+1}^p = \mathrm{P}^{\Sigma_i^p}$.

The superscript "p" distinguishes the polynomial-time hierarchy used in complexity theory from the arithmetic hierarchy from recursion theory (see [Rog87]).

We define the polynomial-time hierarchy (PH) as the union of these classes:

$$\mathrm{PH} = \bigcup_i \Sigma_i^p = \bigcup_i \Pi_i^p = \bigcup_i \Delta_i^p \subseteq \mathrm{PSPACE}.$$

We say that the polynomial-time hierarchy is *infinite* if $\Sigma_i^p \neq \Sigma_{i+1}^p$ for all i. Otherwise, we say that the hierarchy is *finite* or that it *collapses*. Complexity theorists generally believe that the hierarchy is infinite.

3 Counting Functions

In this section, we will discuss the power of counting functions. We will examine two powerful function classes in particular, #P and GapP.

The classes #P (read "Sharp-P" or "Number-P") and GapP are derived directly from looking at the number of accepting paths or the gap between accepting and rejecting paths, respectively:

Definition 3.1 *1. The class* #P *consists of the functions f such that there exists an* NP *machine M such that for all $x \in \Sigma^*$, $f(x) = \#M(x)$.*

2. The class GapP *consists of the functions f such that there exists an* NP *machine M such that for all $x \in \Sigma^*$, $f(x) = \Delta M(x)$.*

First, note that #P and GapP functions cannot take on values too large.

Lemma 3.2 *If $f(x)$ is a* #P *or* GapP *function then there exists a polynomial $p(n)$ such that the absolute value of $f(x)$ is bounded by $2^{p(|x|)}$ for every x.*

Proof: Either $f(x) = \#M(x)$ or $\Delta M(x)$ for some NP machine M. Let $p(n)$ bound the running time of M on strings of length n. Note that $M(x)$ can have at most $2^{p(|x|)}$ computation paths. The numbers of accepting and rejecting paths of $M(x)$ are bounded by the number of computation paths. Lemma 3.2 follows. □

Note that #P functions cannot take on negative values, though GapP functions can. In fact, GapP functions are closed under negation.

Lemma 3.3 *If f is a* GapP *function then $-f$ is also a* GapP *function.*

Proof: Let M be an NP machine such that $f = \Delta M$. Then $-f = \Delta\overline{M}$. □.

How about the relationship between #P and GapP and the polynomial-time computable functions FP? First, we show that every #P function is also a GapP function:

Lemma 3.4 *For every* NP *machine M, there is an* NP *machine N such that $\Delta N = \#M$.*

Proof: Given an input x, the machine N guesses a path p of $M(x)$. If p is accepting, N accepts. Otherwise, N branches once, accepting on one branch and rejecting on the other. We have, for all x,

$$\begin{aligned} \Delta N(x) &= \#N(x) - \#\overline{N}(x) \\ &= \#N(x) - \#\overline{M}(x) \\ &= \#M(x) + \#\overline{M}(x) - \#\overline{M}(x) \\ &= \#M(x), \end{aligned}$$

fulfilling the conclusion of Lemma 3.4. □

Note that a #P function can take on only nonnegative values, so this class cannot capture all of the FP functions. However, it does capture those FP functions that only take on nonnegative values. The class GapP encompasses all the FP functions without restriction.

Theorem 3.5 *If an* FP *function f always takes on nonnegative values then f is in* #P*. Every* FP *function f is in* GapP*.*

Proof: Let f be an FP function. Create an NP machine M that, on input x, guesses $|f(x)|$ computation paths and accepts on these paths if and only if $f(x) > 0$. Note that $\Delta M(x) = f(x)$ for all x and $\#M(x) = f(x)$ when $f(x) \geq 0$. □

It is possible that FP captures all the #P or GapP functions or that #P captures all the GapP functions that take on only nonnegative values, but these propositions seem unlikely. However, from the following lemma, we will see that #P and GapP have essentially the same computational power.

Lemma 3.6 *For all functions f, the following are equivalent:*

1. $f \in$ GapP.
2. *f is the difference of two* #P *functions.*
3. *f is the difference of a* #P *function and an* FP *function.*
4. *f is the difference of an* FP *function and a* #P *function.*

Corollary 3.7 $\text{GapP} \subseteq \text{FP}^{\#\text{P}[1]}$.

Proof: Follows from the third characterization of GapP in Lemma 3.6. □

Proof of Lemma 3.6:

$(1 \Rightarrow 2)$ For any M we have

$$\Delta M = \#M - \#\overline{M}$$

by definition, so GapP is the difference of two #P functions.

$(2 \Rightarrow 3)$ Let f and g be #P functions. We can assume that $f = \#M$ and $g = \#N$, where M and N are NP machines, and N has $2^{q(n)}$ computation paths for some polynomial q (just pad N with extra rejecting paths so that all paths have length $q(n)$). Let M' be the machine that first branches once, then simulates M on one branch and $\overline{N}$ on the other. We have, for any x,

$$\begin{aligned} f(x) - g(x) &= \#M(x) - \#N(x) \\ &= \#M(x) + \#\overline{N}(x) - 2^{q(|x|)} \\ &= \#M'(x) - 2^{q(|x|)}. \end{aligned}$$

Therefore $f - g$ is the difference of a #P and an FP function.

$(3 \Rightarrow 1)$ Let f be a #P function and let $g \in$ FP. Let M be the NP machine such that $f = \Delta M$. Let N be such that, for all $x \in \Sigma^*$, $N(x)$ resembles $M(x)$ padded with $g(x)$ rejecting paths. We then have $\Delta N = f - g$.

$(3 \Leftrightarrow 4)$ This follows since we now have $(1 \Leftrightarrow 3)$, and the GapP functions are closed under negation. □

The composition of two FP functions remains an FP function. However, it seems unlikely that the composition of two #P functions is a #P or even a GapP function. We can, however, combine #P and GapP functions with FP functions.

Lemma 3.8 *Let f be an* FP *function and g a* #P *or* GapP *function. Then $g(f(x))$ is a* #P *or* GapP *function, respectively.*

Proof: Let M be an NP machine that defines a #P function g. Define N to be an NP machine that on input x simulates $M(f(x))$. Then $\#N(x)$ is exactly $g(f(x))$. The proof for GapP functions is identical. □

3.1 Algebraic Properties of Counting Functions

The vast power of #P and GapP functions arises from their closure under algebraic operations like exponential summation and polynomial products.

Lemma 3.9 *Let f be a* #P *function and q a polynomial. Then, the following are also* #P *functions:*

1. $\sum_{|y| \leq q(|x|)} f(x, y)$, *and*
2. $\prod_{0 \leq y \leq q(|x|)} f(x, y)$.

Proof: Fix an NP machine M such that $f(x, y) = \#M(x, y)$.

1. Define an NP machine N that on input x guesses a string y of length bounded by $q(|x|)$ and then simulates $M(x, y)$.

2. Notice that if one simulates two nondeterministic machines and accepts if both accept, then the number of accepting paths of the new machine will be exactly the product of the numbers of accepting paths of the original two machines. To prove part 2, we generalize this idea.

   Define an NP machine N that on input x simulates the following:

   FOR y from 0 to $q(|x|)$
   Simulate $M(x, y)$
   If $M(x, y)$ rejects then REJECT
   ACCEPT

In each case, $N(x)$ will have the appropriate number of accepting computations. □

Lemma 3.10 *Let f be a* GapP *function and q a polynomial. Then, the following are* GapP *functions:*

1. $\sum_{|y|\leq q(|x|)} f(x,y)$, *and*
2. $\prod_{0\leq y\leq q(|x|)} f(x,y)$.

Proof: Fix an NP machine M such that $f(x,y) = \Delta M(x,y)$.

1. Same as in the proof of Lemma 3.9.
2. Let $g(x) = \prod_{0\leq y\leq q(|x|)} f(x,y)$. Define an NP machine N that on input x guesses, in sequence, computation paths of M on the inputs $\langle x,0\rangle$, $\langle x,1\rangle$, $\langle x,2\rangle$, and so on through $\langle x,q(|x|)\rangle$. N accepts if an even number of these paths are rejecting, and N rejects if an odd number of these paths are rejecting. The machine N is an NP machine. The fact that $g = \Delta N$ can be shown by induction on the value $n = q(|x|)$ as follows: For $n = 0$, we have $\Delta N(x) = f(x,0) = g(x)$ because $N(x)$ behaves just as $M(x,0)$ does. If $n > 0$, assume that the inductive hypothesis is true for $n-1$, and let N' be a machine that acts the same as N except that N' guesses paths of M only on inputs $\langle x,0\rangle,\ldots,\langle x,n-1\rangle$. For convenience, let $a_{N'} = \#N'(x)$, $r_{N'} = \#\overline{N'}(x)$, $a_M = \#M(x,n)$, and $r_M = \#\overline{M}(x,n)$. By the inductive hypothesis, we have

$$\begin{aligned} g(x) &= \Delta N'(x) f(x,n) \\ &= \Delta N'(x) \Delta M(x,n) \\ &= (a_{N'} - r_{N'})(a_M - r_M) \\ &= (a_{N'}a_M + r_{N'}r_M) - (a_{N'}r_M + r_{N'}a_M). \end{aligned}$$

Now $N(x)$ accepts whenever it guesses an even number of rejecting paths. This happens either when there are an even number of rejections through $\langle x,n-1\rangle$ and the last path is accepting, or when there are an odd number of rejections through $\langle x,n-1\rangle$ and the last path is rejecting. Thus by the definition of N', the total number of sequences accepted by $N(x)$ is exactly $a_{N'}a_M + r_{N'}r_M$. Likewise, the total number of sequences rejected by $N(x)$ is $a_{N'}r_M + r_{N'}a_M$. Therefore, $g(x) = \Delta N(x)$. □

With the closure properties of Lemma 3.9 and 3.10, one would expect a close connection between #P and GapP functions and low-degree polynomials. Babai and Fortnow [BF91] show that in fact #P and GapP functions can be expressed as polynomials built up in a simple way.

A *retarded arithmetic program with binary substitutions* (RAB) will be a sequence $\{p_1, p_2, \ldots\}$ of instructions such that for every k, one of the following holds:

(1) p_k is one of the constant polynomials 0 or 1;

(2) $p_k = x_i$ for some $i \leq k$;

(3) $p_k = 1 - x_i$ for some $i \leq k$;

(4) $p_k = p_i - p_j$ for some $i, j < k$;

(5) $p_k = p_i p_j$ for some i, j such that $i + j \leq k$ *(retarded multiplication);*

(6) $p_k = p_j(x_i = 0)$ or $p_j(x_i = 1)$ for some $i, j < k$ *(binary substitution).* (Here, $p_j(x_i = \varepsilon)$ refers to the polynomial obtained from p_j by replacing the variable x_i by the value ε.)

We call a RAB uniform if there exists a polynomial-time function that on input 1^n outputs the first n instructions.

Theorem 3.11 ([BF91, FFK94]) *For every function f the following are equivalent:*

1. $f \in$ GapP.
2. *There exists a uniform* RAB *and a polynomial $q(n)$ such that $p_{q(n)}$ has $x_1, \ldots, x_n$ as free variables and such that for every string $x \in \{0,1\}^*$, $f(x) = p_{q(n)}(x)$.*

3.2 A Randomized sign *Function*

Valiant and Vazirani [VV86] show how to randomly "isolate" one assignment from a satisfying assignment. In the context of counting functions, their construction in some sense allows us to apply a "randomized" *sign* function to #P functions.

Lemma 3.12 (Valiant-Vazirani) *Let f be a* #P *function. There exist a* #P *function g and polynomials q and t such that*

1. *If $f(x) = 0$ then $g(x, r) = 0$ for all $r \in \Sigma^{q(n)}$.*
2. *If $f(x) > 0$ then*

$$\Pr_{r \in \Sigma^{q(n)}}(g(x, r) = 1) \geq \frac{1}{t(n)}.$$

In order to prove Lemma 3.12, Valiant and Vazirani [VV86] and Mulmuley, Vazirani, and Vazirani [MVV87] use probabilistic techniques. We give a new proof based on the algebraic techniques of Buhrman and Fortnow [BF95]. These techniques are related to the "designs" of Nisan [Nis91] and Nisan and Wigderson [NW94].

Proof: Let M be an NP machine such that $f(x) = \#M(x)$. Fix x and suppose $f(x) > 0$. Let ℓ be the length of a binary encoding of a computation path of $M(x)$. Let S be the set of accepting computation paths of $M(x)$ encoded as strings of length ℓ, and let $d = |S| = f(x) = \#M(x)$.

Pick m such that $2\ell d < 2^m \leq 4\ell d$. Let $F = \mathrm{GF}(2^m)$, the finite field of 2^m elements.

We will encode the computation paths as polynomials over F. We then consider pairs $(a, b) \in F^2$ and show that for a sizable fraction of them there will be exactly one polynomial p representing an accepting path of M such that $p(a) = b$. Lemma 3.12 will follow by choosing m, a, and b at random.

For a string $y = y_1 \ldots y_\ell$ in S, consider the following ℓ-degree polynomial over F:

$$p_y(X) = \sum_{i=1}^{\ell} y_i X^i.$$

Fix a y in S. An element a of F will be called y-good if, for all $z \neq y$ in S, $p_y(a) \neq p_z(a)$. Since p_y and p_z can agree on at most ℓ elements of F, there are at least $|F| - \ell d$ y-good elements in F.

Consider the set A_y defined as the set of pairs $(a, p_y(a))$ for all y-good a. Note that for two different y and z in S, $A_y \cap A_z = \emptyset$. Let $A = \cup_{y \in S} A_y$. Note that $|A| \geq d(|F| - \ell d)$.

Now we can define the #P function g by defining the corresponding NP machine N. The machine N on input $\langle x, r \rangle$ does the following.

- Use r as an encoding of an integer m^* between 1 and 2ℓ and elements a and b in $\mathrm{GF}(2^{m^*})$.
- Guess a string y of length ℓ and accept if both: (a) $p_y(a) = b$ in the field $\mathrm{GF}(2^{m^*})$, and (b) y encodes an accepting computation of $M(x)$.

Shoup [Sho90] shows how to find an irreducible polynomial over GF(2) of degree m^* in polynomial time. One can use this irreducible polynomial to do field operations in $\mathrm{GF}(2^{m^*})$ in polynomial time.

If $f(x) = 0$ then $g(x, r) = 0$ for all r, since $N(x, r)$ will never accept.

If $f(x) > 0$ note that $d \leq 2^\ell$ so $m \leq \log 4d\ell < 2l$. With probability $1/(2\ell)$ we have $m = m^*$.

Assume that m^* was chosen to be m. Note that if (a, b) is in A then $N(x, r)$ has exactly one accepting computation, and thus $g(x, r) = 1$ as desired.

We have that the size of A is at least $d(|F| - \ell d) \geq d(2\ell d - \ell d) = \ell d^2$.

We also have that the size of F^2 is at most $16\ell^2 d^2$.

Thus if we choose (a, b) at random in F^2 we have at least a $1/(16\ell)$ chance of being in A.

Fix x. The probability of choosing r at random such that $m = m^*$ and (a, b) in A is at least $1/(32\ell^2)$.

Thus the probability that $N(x, r)$ has exactly one accepting computation (and $g(x, r) = 1$) is at least $1/(32\ell^2)$. Since ℓ is bounded by a polynomial in $|x|$, we have fulfilled condition 2 of Lemma 3.12. □

While the $1/t(n)$ term in Lemma 3.12 does not seem like a large probability, one can take many samples of g and have an extremely high likelihood of having one of them take on a value of one.

Corollary 3.13 *Let f be a* #P *function and q a polynomial. There exist a* #P *function g and polynomials p and t such that*

1. *If $f(x) = 0$ then $g(x, i, r) = 0$ for all i such that $1 \leq i \leq t(n)$ and $r \in \Sigma^{p(n)}$.*

2. *If $f(x) > 0$ then*

$$\Pr_{r \in \Sigma^{p(n)}}(\exists i \ 1 \leq i \leq t(n) \text{ such that } g(x, i, r) = 1) \geq 1 - 2^{-q(n)}.$$

It does not seem likely that a variant of Lemma 3.12 or Corollary 3.13 holds for GapP functions.

3.3 Counting Functions and the Polynomial-Time Hierarchy

Toda and Ogihara [TO92] show that the polynomial-time hierarchy does not add any power to GapP functions in a probabilistic sense.

Theorem 3.14 *Let f be in* $\mathrm{GapP}^{\mathrm{PH}}$ *and let p be a polynomial. Then there exist a two-argument function g in* GapP *and a polynomial q such that for all x*

$$\Pr_{r \in \Sigma^{q(n)}}(g(x, r) = f(x)) \geq 1 - 2^{-p(n)}.$$

In order to prove this theorem we use the following lemma proven later in this section.

Lemma 3.15 *Let f be in* $\#\mathrm{P}^{\mathrm{NP}}$ *and p be a polynomial. There exist a two-argument function g in* GapP *and a polynomial q such that for all x*

$$\Pr_{r \in \Sigma^{q(n)}}(g(x, r) = f(x)) \geq 1 - 2^{-p(n)}.$$

Proof of Theorem 3.14: By induction, we will show that Theorem 3.14 holds for all f in $\mathrm{GapP}^{\Sigma_k^p}$. The base case $k = 0$ is trivial since $\mathrm{GapP}^{\Sigma_0^p} = \mathrm{GapP}^{\mathrm{P}} = \mathrm{GapP}$.

Assume that the inductive hypothesis holds for k. Let f be in $\mathrm{GapP}^{\Sigma_{k+1}^p}$ and fix a polynomial p. Let A be a Σ_k^p-complete set. We have f in $\mathrm{GapP}^{\mathrm{NP}^A}$. By the fact that Lemma 3.6 relativizes, there exist f^+ and f^- in $\#\mathrm{P}^{\mathrm{NP}^A}$ such that $f = f^+ - f^-$. Since the proof of Lemma 3.15 also relativizes, let $g_1^+(x, r_1)$ and $g_1^-(x, r_1)$ in GapP^A be the functions fulfilling the conclusion of Lemma 3.15 with polynomial $p + 2$ for $f^+(x)$ and $f^-(x)$, respectively.

Apply the inductive hypothesis to the functions $g_1^+(x, r_1)$ and $g_1^-(x, r_1)$ to get $g_2^+(\langle x, r_1\rangle, r_2)$ and $g_2^-(\langle x, r_1\rangle, r_2)$ in GapP, fulfilling the conclusion of Theorem 3.14 for polynomial $p + 2$.

We now combine g_2^+ and g_2^- to get the final g that fulfills Theorem 3.14 for f. Define g by

$$g(x, r_1 r_2) = g_2^+(\langle x, r_1\rangle, r_2) - g_2^-(\langle x, r_1\rangle, r_2).$$

Fix an input x. The following events occur each with probability at least $1 - 2^{-(p(n)+2)}$ when r_1 and r_2 are chosen at random:

1. $g_1^+(x, r_1) = f^+(x)$.
2. $g_1^-(x, r_1) = f^-(x)$.
3. $g_2^+(\langle x, r_1\rangle, r_2) = g_1^+(x, r_1)$.
4. $g_2^-(\langle x, r_1\rangle, r_2) = g_1^-(x, r_1)$.

Thus with probability at least $1 - 2^{p(n)}$ we have

$$f(x) = f^+(x) - f^-(x) = g_2^+(\langle x, r_1\rangle, r_2) - g_2^-(\langle x, r_1\rangle, r_2) = g(x, r_1 r_2),$$

fulfilling the inductive hypothesis for $k + 1$. □

We will need the following lemma for the proof of Lemma 3.15.

Lemma 3.16 *For every f in $\#\mathrm{P}^{\mathrm{NP}}$ there is a language A in* coNP *and a polynomial r such that*

$$f(x) = ||y \in \Sigma^{r(n)} : \langle x, y\rangle \in A||.$$

Proof: Let M be the NP machine such that $f = \#M^{\mathrm{SAT}}$. Fix an input x. Let y represent the following tuple:

$$\langle p, s_1, \ldots, s_q\rangle,$$

where p represents a computation path of $M^{\mathrm{SAT}}(x)$, q is the number of oracle queries made on computation path p, and s_i is either ϵ or a boolean assignment.

Define A as the set of tuples $\langle x, y\rangle$ such that p is an accepting path of $M(x)$ and if ϕ is the ith query made by $M(x)$ on computation path p either

1. ϕ is not in SAT and $s_i = \epsilon$, or
2. ϕ is in SAT and s_i is the lexicographically least satisfying assignment to ϕ.

For each input x there is exactly one y with $\langle x, y\rangle$ in A for each accepting computation of $M^{\mathrm{SAT}}(x)$. The set A is in coNP since in (2.) we need only check that s_i is an assignment to ϕ and that no $s < s_i$ are assignments to ϕ. □

Proof of Lemma 3.15: Let f be a function in $\#\mathrm{P}^{\mathrm{NP}}$ and let A and $r(n)$ be as derived from Lemma 3.16.

Fix a polynomial p. By Corollary 3.13, there is a #P function $f_1(i,w,x,y)$ and a polynomial q_1, where i ranges from 1 to $q_1(n)$ and w ranges over strings of length $q_1(n)$, such that

1. If $\langle x,y\rangle$ is not in A then $f_1(i,w,x,y)=0$, and
2. if $\langle x,y\rangle$ is in A then

$$\Pr_w(\exists i\ f_1(i,w,x,y)=1) \geq 1-2^{-r(n)p(n)}.$$

Define the GapP function f_2 by

$$f_2(w,x,y) = 1 - \prod_{i=1}^{q(n)} (1 - f_1(i,w,x,y)).$$

Note that if (x,y) is not in A then for all w, $f_2(w,x,y)=0$. If on the other hand (x,y) is in A then with probability at least $1-2^{-r(n)p(n)}$, $f_2(w,x,y)=1$.

Finally, we define the GapP function $g(x,w)=\sum_y f_2(w,x,y)$. The probability that $g(x,w)\neq f(x)$ is at most $2^{-r(n)p(n)}$ times the number of y's $(2^{r(n)})$, for a total probability bounded by $2^{-p(n)}$. □

4 Counting Classes

We can use counting functions to define language classes in a natural way. We can then use the closure properties of #P and GapP functions described in Section 3 to help us understand the complexity of these language classes.

4.1 Classifying Counting Classes

Let L be a language in NP accepted by an NP machine M. If we look at the #P function $f(x)=\#M(x)$, note that $f(x)>0$ exactly when x is in L.

This works both ways: Suppose that we have a #P function f defined by an NP machine M. If we look at the set of strings x such that $f(x)>0$, this is exactly the NP set accepted by M.

Thus we have the following characterization of the class NP using #P functions:

Classification 4.1 *The class* NP *consists of those languages L such that for some* #P *function f and all x in Σ^**

- *If x is in L then $f(x)>0$.*

- *If x is not in L then $f(x) = 0$.*

In fact, many classes have natural and simple characterizations using #P and GapP functions. Often these new characterizations allow us to use the closure properties discussed in Section 3 to prove results about these classes. We use the term "counting classes" to refer to complexity classes with "simple" characterizations using #P or GapP functions.

The class UP ("Unique P") consists of those NP languages accepted by NP machines that never have more than one accepting path. In terms of #P functions:

Classification 4.2 *The class* UP *consists of those languages L such that for some* #P *function f and all x in Σ^**

- *If x is in L then $f(x) = 1$.*
- *If x is not in L then $f(x) = 0$.*

Gill [Gil77] defined the class PP ("Probabilistic Polynomial Time") as the set of languages L with probabilistic polynomial-time Turing machines M where x is in L if the probability of $M(x)$ accepting is greater than one-half. If one considers M as a nondeterministic machine, this means that the accepting paths outnumber the rejecting paths. In terms of GapP functions:

Classification 4.3 *The class* PP *consists of those languages L such that for some* GapP *function f and all x in Σ^**

- *If x is in L then $f(x) > 0$.*
- *If x is not in L then $f(x) < 0$.*

As complexity theorists discovered these simple characterizations, several other classes were defined using #P and GapP functions:

Classification 4.4 *The class* SPP *consists of those languages L such that for some* GapP *function f and all x in Σ^**

- *If x is in L then $f(x) = 1$.*
- *If x is not in L then $f(x) = 0$.*

Classification 4.5 *The class* $C_=P$ *consists of those languages L such that for some* GapP *function f and all x in Σ^**

- *If x is in L then $f(x) = 0$.*
- *If x is not in L then $f(x) \neq 0$.*

Classification 4.6 *The class* $\oplus$P *consists of those languages L such that for some* #P *function f and all x in Σ^**

- *If x is in L then $f(x)$ is odd.*
- *If x is not in L then $f(x)$ is even.*

Classification 4.7 *The class* $\mathrm{Mod}_k\mathrm{P}$ *consists of those languages L such that for some* #P *function f and all x in Σ^**

- *If x is in L then $f(x) \bmod k \neq 0$.*
- *If x is not in L then $f(x) \bmod k = 0$.*

Note that $\oplus$P is just Mod_2P. Fenner, Fortnow, and Kurtz [FFK94] show that we can replace #P with GapP in Classifications 4.6 ($\oplus$P) and 4.7 (Mod_kP).

We can sometimes simplify some of these characterizations using some of the closure properties discussed in Section 3. For example, if f is a GapP function then the GapP function $f^2(x) > 0$ exactly when $f(x) \neq 0$. This allows us to have a new characterization of $\mathrm{C}_=\mathrm{P}$:

Classification 4.8 *The class* $\mathrm{C}_=\mathrm{P}$ *consists of those languages L such that for some* GapP *function f and all x in Σ^**

- *If x is in L then $f(x) = 0$.*
- *If x is not in L then $f(x) > 0$.*

If a class $\mathcal{C}$ has a characterization using counting functions then co-$\mathcal{C}$ also has a characterization using counting functions where we reverse the implicants. For example:

Classification 4.9 *The class* co-$\mathrm{C}_=\mathrm{P}$ *consists of those languages L such that for some* GapP *function f and all x in Σ^**

- *If x is in L then $f(x) > 0$.*
- *If x is not in L then $f(x) = 0$.*

From Classifications 4.3 and 4.9, we get

Corollary 4.10 $\text{co-}\mathrm{C}_=\mathrm{P} \subseteq \mathrm{PP}$.

Also, since every #P function is a GapP function, we have

Corollary 4.11

- $\mathrm{NP} \subseteq \text{co-}\mathrm{C}_=\mathrm{P}$.
- $\mathrm{UP} \subseteq \mathrm{SPP}$.

For a GapP function f, the GapP function $2f(x) - 1$ has the same sign as $f(x)$ for positive and negative $f(x)$ and takes on the value -1 for $f(x) = 0$. Thus we have a subtle but important variation of Classification 4.3:

Classification 4.12 *The class* PP *consists of those languages L such that for some* GapP *function f and all x in Σ^**

- *If x is in L then $f(x) > 0$.*
- *If x is not in L then $f(x) < 0$.*

From Classification 4.12 and the fact that GapP functions are closed under negation, we have the following.

Corollary 4.13

- PP *is closed under complement.*
- $\mathrm{C}_{=}\mathrm{P} \subseteq \mathrm{PP}$.

We can also use these classifications to show a relationship between PP, #P, and GapP.

Theorem 4.14 $\mathrm{P}^{\mathrm{PP}} = \mathrm{P}^{\mathrm{GapP}}$.

Proof: We only need to show that every GapP function g is computable in $\mathrm{FP}^{\mathrm{PP}}$. Consider the language

$$L = \{\langle x, k\rangle \mid g(x) > k\}.$$

If we consider the GapP function $f(x,k) = g(x) - k$, we have that $L \in \mathrm{PP}$ by Classification 4.3. One can then use binary search using L as an oracle to find the value of $g(x)$. □

In one sense, PP languages consider the high-order bit of a GapP function, and $\oplus$P languages consider the low-order bit. Green, Köbler, Regan, Schwentick, and Torán [GKR+95] looked at the class of languages that consider the middle bit.

Consider the function *Middle* that on input x returns the $\lceil \frac{|x|}{2} \rceil$th bit of the string x. Remember that we also consider integers as strings as defined in Section 2. We can then define the class MP using our usual classification.

Classification 4.15 *The class* MP *consists of those languages L such that for some* GapP *function f and all x in Σ^**

- *If x is in L then* $\mathrm{Middle}(x) = 1$.
- *If x is not in L then* $\mathrm{Middle}(x) = 0$.

Note: Green et al. define the class MP using an equivalent definition that employs #P functions and a polynomial-time computable pointer to the particular bit in question. By appropriately padding the #P function, one can use the middle bit to determine any other single bit. Also, one can easily convert a GapP function to a #P function without modifying the middle bit.

We have that MP contains $\oplus$P and PP. We also have $\mathrm{MP} \subseteq \mathrm{P}^{\#\mathrm{P}[1]}$. As we will see in Corollary 4.22, MP actually contains the entire polynomial-time hierarchy.

4.2 Counting Operators

Given a set A and a polynomial p, we can define the function $\#_A^p$ as follows:

$$\#_A^p(x) = |\{y \in \Sigma^{p(|x|)} \mid \langle x, y\rangle \in A\}|.$$

Varying the complexity of A allows us to define other complexity classes. We use this idea to define counting operators.

An operator maps one complexity class to another. For example, we define the class $\# \cdot \mathcal{C}$ as follows:

Definition 4.16 *$\# \cdot \mathcal{C}$ consists of the set of functions f such that for some $A \in \mathcal{C}$ and some polynomial p we have for all x, $f(x) = \#_A^p(x)$.*

We can define the operator $\mathrm{Gap} \cdot \mathcal{C}$ as the difference between two $\# \cdot \mathcal{C}$ functions. Note that $\# \cdot \mathrm{P} = \#\mathrm{P}$ and $\mathrm{Gap} \cdot \mathrm{P} = \mathrm{GapP}$.

We can also use counting operators to generate language classes as well as function classes by generalizing the classifications in Section 4.1.

Classification 4.17 *The class $\mathrm{P} \cdot \mathcal{C}$ consists of those languages L such that for some $\mathrm{Gap} \cdot \mathcal{C}$ function f and all x in Σ^**

- *If x is in L then $f(x) > 0$.*
- *If x is not in L then $f(x) \leq 0$.*

In fact, one can generalize all of the classifications in Section 4.1 in a similar way. For further details and a history of counting operators, see the survey by Hemaspaandra and Vollmer [HV95].

4.3 The Polynomial-Time Hierarchy

We can combine the results in Sections 3.3 and 4.2 to help us understand the relationship between counting classes and the polynomial-time hierarchy.

We will now complete the proof of the following beautiful result of Seinosuke Toda [Tod91].

Theorem 4.18

$$\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{GapP}[1]}.$$

By Corollary 3.7 and Theorem 4.14, we immediately get the following corollaries.

Corollary 4.19

1. $\mathrm{PH} \subseteq \mathrm{P}^{\#\mathrm{P}[1]}$.
2. $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{PP}}$.

First, we prove some lemmas relating the polynomial-time hierarchy to counting operators on counting classes.

Lemma 4.20

1. PH $\subseteq$ P $\cdot$ PP.
2. PH $\subseteq$ P $\cdot \oplus$P.
3. PH $\subseteq$ P $\cdot$ C$_=$P.

Proof: We will prove only the second item of Lemma 4.20. The other two have similar proofs.

Fix a language L in PH. Consider the characteristic function χ_L that takes on the value one for $x \in L$ and zero otherwise.

Since $\chi_L \in \mathrm{GapP}^{\mathrm{PH}}$, we can apply Theorem 3.14 to get a GapP function g such that

$$\Pr_{r \in \Sigma^{q(n)}}(g(x,r) = \chi_L(x)) \geq \frac{3}{4},$$

and thus for every $x \in \Sigma^*$,

1. If $x \in L$ then
$$|\{r \in \Sigma^{q(n)} \mid g(x,r) = 1\}| > |\{r \in \Sigma^{q(n)} \mid g(x,r) \neq 1\}|.$$
2. If $x \notin L$ then
$$|\{r \in \Sigma^{q(n)} \mid g(x,r) = 0\}| > |\{r \in \Sigma^{q(n)} \mid g(x,r) \neq 0\}|.$$

Consider the set A consisting of $\langle x, r\rangle$ such that $g(x,r)$ is odd. Since $g(x,r)$ is in GapP, we have A in $\oplus$P. Let f be the GapP function defined by $\#^p_A(x) - \#^p_{\overline{A}}(x)$. We have that L is in P $\cdot \oplus$P by Classification 4.17. □

A couple of notes about Lemma 4.20:

1. One can replace P$\cdot$ with BP$\cdot$ in each item of Lemma 4.20, where BP$\cdot$ represents a bounded error version of P$\cdot$. We refer the interested reader to the paper of Toda and Ogihara [TO92] for more details.
2. The statement "PH $\subseteq$ P $\cdot$ SPP" does not follow directly using the techniques of the proof of Lemma 4.20. Though for most r, $g(x,r) \in \{0,1\}$, there may be some r where $g(x,r)$ takes on other values.

Proof of Theorem 4.18: Fix L in PH. By Lemma 4.20, we have L in P $\cdot \oplus$P. In other words, there exist a GapP function g and a polynomial q such that x is in L if and only if

$$|\{r \in \Sigma^{q(n)} \mid g(x,r) \bmod 2 = 1\}| > |\{r \in \Sigma^{q(n)} \mid g(x,r) \bmod 2 = 0\}|.$$

Let $\mathcal{R}_1$ and $\mathcal{R}_0$ represent these two sets, respectively.

While this function g does not directly lead to a proof of Theorem 4.18, we can use g to create a new GapP function $\hat{g}$ with more useful properties.

Lemma 4.21 *For every polynomial p, there exists a* GapP *function $\hat{g}$ such that for all x and r,*

1. *If $g(x,r) \bmod 2 = 1$ then $\hat{g}(x,r) \bmod 2^{p(n)} = 1$, and*
2. *if $g(x,r) \bmod 2 = 0$ then $\hat{g}(x,r) \bmod 2^{p(n)} = 0$.*

First, we show how to use Lemma 4.21 to finish the proof of Theorem 4.18. Let $p(n) = q(n) + 1$, and let $\hat{g}$ be the result of applying Lemma 4.21 to g.

Consider the GapP function h defined by

$$h(x) = \sum_{r \in \Sigma^{q(n)}} \hat{g}(x,r).$$

Note that

$$\begin{aligned} h(x) \bmod 2^{p(n)} &= (\sum_{r \in \Sigma^{q(n)}} \hat{g}(x,r)) \bmod 2^{p(n)} \\ &= (\sum_{r \in \mathcal{R}_1} \hat{g}(x,r)) \bmod 2^{p(n)} + (\sum_{r \in \mathcal{R}_0} \hat{g}(x,r)) \bmod 2^{p(n)} \\ &= \sum_{r \in \mathcal{R}_1} (\hat{g}(x,r) \bmod 2^{p(n)}) + \sum_{r \in \mathcal{R}_0} (\hat{g}(x,r) \bmod 2^{p(n)}) \\ &= |\mathcal{R}_1| \bmod 2^{p(n)} \\ &= |\mathcal{R}_1|, \end{aligned}$$

since $|\mathcal{R}_1| \leq 2^{q(n)} < 2^{p(n)}$.

We have that x is in L if and only if $h(x) \bmod 2^{p(n)} > 2^{q(n)}/2$. We can thus determine whether x is in L by a single query to the GapP function h. □

We do not need the entire value of $h(x)$. We have that x is in L if and only if the $q(n)$th bit from the right of $h(x)$ is one. Thus we get the following corollary about the surprising power of MP.

Corollary 4.22 PH $\subseteq$ P $\cdot \oplus$P $\subseteq$ MP.

Proof of Lemma 4.21: Consider the innocuous looking polynomial $f(m) = 3m^2 - 2m^3$. This polynomial has the following easily verifiable properties for every positive integer m and i.

1. If $m \bmod 2^i = 1$ then $f(m) \bmod 2^{2i} = 1$.
2. If $m \bmod 2^i = 0$ then $f(m) \bmod 2^{2i} = 0$.

Consider $f^{(k)}$, the function f iterated k times. We have

1. If $m \bmod 2 = 1$ then $f^{(k)}(m) \bmod 2^{2^k} = 1$.
2. If $m \bmod 2 = 0$ then $f^{(k)}(m) \bmod 2^{2^k} = 0$.

Define $\hat{g}(x,r) = f^{(\lceil \log_2 p(n) \rceil)}(g(x,r))$. By the property of f, $\hat{g}$ fulfills the conditions required by Lemma 4.21.

We need to argue that $\hat{g}$ is a GapP function. Note that $f^{(k)}(m)$ is a polynomial in m of degree 3^k. The degree of $f^{(\lceil \log_2 p(n) \rceil)}$ is bounded by $3^{1+\log_2 p(n)} = 3 \cdot 3^{\log_2 p(n)} = 3 \cdot (2^{\log_2 3})^{\log_2 p(n)} = 3p(n)^{\log_2 3} < 3p(n)^2$—still a polynomial in n. Thus by appropriately applying Lemma 3.10 we have that $\hat{g}$ is a GapP function. □

Note: Toda [Tod91] proved a version of Lemma 4.21 using a different f. We find it simpler to use the function f given by Yao [Yao90]. See Beigel and Tarui [BT94] for an in-depth look at these modulus-amplifying polynomials.

4.4 Closure Properties of PP

In his original paper on probabilistic complexity classes, Gill [Gil77] showed that PP is closed under complement (Corollary 4.13). Gill left open the question as to whether PP is closed under union. Note that by DeMorgan's Law, since PP is closed under complement, closure under union and closure under intersection are equivalent questions.

This question remained open for many years until Beigel, Reingold, and Spielman [BRS95] showed that in fact the class PP is closed under union and thus intersection. In this section, we will present the basic idea of that proof.

Beigel, Reingold, and Spielman, extending work of Newman [New64], developed some rational functions that closely approximated the *sign* function. (A rational function is a quotient of two polynomials.)

Let m be an even positive integer. Based on similar polynomials due to Beigel, Reingold, and Spielman, we define

$$\begin{aligned} P_m(z) &= (z-1)\prod_{i=1}^{m}(z-2^i)^2. \\ A_m &= (P_m(-z))^{\frac{m}{2}+1} - (P_m(z))^{\frac{m}{2}+1}. \\ B_m &= (P_m(-z))^{\frac{m}{2}+1} + (P_m(z))^{\frac{m}{2}+1}. \\ S_m &= \frac{A_m}{B_m}. \end{aligned}$$

Lemma 4.23 ([BRS95])

1. *If* $1 \le z \le 2^m$ *then* $1 \le S_m < 1 + 2^{-m}$.
2. *If* $-2^m \le z \le -1$ *then* $-1 - 2^{-m} < S_m \le -1$.

Note that if f is a GapP function and m is a polynomial in $|x|$ with even coefficients, we have that $A_m \circ f$ and $B_m \circ f$ are also GapP computable functions. This follows from the closure properties in Section 3.

We now use Lemma 4.23 to show the following main result from the paper of Beigel, Reingold, and Spielman [BRS95]:

Theorem 4.24 *The class* PP *is closed under union.*

Proof: Fix two languages D and E in PP. We will show that $D \cup E$ is also in PP.

Let f_D and f_E be the functions given by Classification 4.12 for D and E, respectively. By Lemma 3.2, let $m = m(|x|)$ be a polynomial with even coefficients such that 2^m bounds the maximum absolute value of $f_D(x)$ and $f_E(x)$. Define $A_D(x) = A_m(f_D(x))$. Similarly, define B_D, S_D, A_E, B_E, and S_E. Note that A_D, B_D, A_E, and B_E are GapP functions.

Let $H = S_D + S_E + 1$. By Lemma 4.23, we have

Lemma 4.25

1. *If* $x \in D$ *and* $x \in E$ *then* $H(x) \geq 3$.
2. *If* $x \in D$ *and* $x \notin E$ *then* $H(x) \geq 1 - 2^{-m}$.
3. *If* $x \notin D$ *and* $x \in E$ *then* $H(x) \geq 1 - 2^{-m}$.
4. *If* $x \notin D$ *and* $x \notin E$ *then* $H(x) \leq -1$.

Thus we have that $x \in D \cup E$ if and only if $H(x) > 0$.

We would be finished if H were a GapP function, but unfortunately it may even take on nonintegral values. We do have

$$H = \frac{A_D}{B_D} + \frac{A_E}{B_E} + 1 = \frac{A_D B_E + A_E B_D + B_D B_E}{B_D B_E}.$$

Note that for nonzero integers p and q we have $p/q > 0$ if and only if $pq > 0$. Thus we define

$$H' = (A_D B_E + A_E B_D + B_D B_E)(B_D B_E).$$

We have that

1. H' is a GapP function.
2. For all $x \in \Sigma^*$, $H(x) > 0$ if and only if $H'(x) > 0$.
3. $x \in D \cup E$ if and only if $H'(x) > 0$.

Finally, applying Classification 4.12 to H', we have $D \cup E \in$ PP. □

Beigel, Reingold, and Spielman [BRS95] also showed several stronger closure properties. Fortnow and Reingold [FR96] extend the work of Beigel, Reingold, and Spielman to show that PP is closed under truth-table reduction and in fact constant-round truth-table reductions. However, showing that PP is closed under Turing reductions would require some nonrelativizing proof techniques (see Section 5).

5 Relativization

The results in this paper show that one complexity class contains another or that some complexity classes have certain closure properties. One would also hope to see separation results like a proof showing, for example, that $\mathrm{PP} \neq \oplus\mathrm{P}$.

The current tools in computational complexity theory do not allow us to achieve this goal. In fact, it could be that $\mathrm{P} = \mathrm{PSPACE}$. Such a circumstance would make most of the results in this paper quite trivial and would collapse all the counting classes in Section 4.1. While we do not believe that $\mathrm{P} = \mathrm{PSPACE}$, we cannot rule out this possibility with known techniques.

Recall the definitions of oracle Turing machines and relativized complexity classes defined in Section 2. Besides using oracles to define complexity classes, we can also use oracles to help us understand what complexity problems may require new techniques.

Consider a relativized version of Theorem 4.24.

Theorem 5.1 *For all oracles A, the class PP^A is closed under union.*

Proof: Fix an oracle A. If we analyze the proof of Theorem 4.24 allowing all Turing machines to have access to A as an oracle, the proof goes through without additional change. □

In fact, all of the proofs in this survey have this relativization property. Suppose that we had a hypothesis like $\mathrm{P} = \mathrm{NP}$. If we can create a relativized world (an oracle A) where $\mathrm{P}^A \neq \mathrm{NP}^A$, then to prove $\mathrm{P} = \mathrm{NP}$ would require techniques that do not relativize, of which we know few. If we also could create an oracle B such that $\mathrm{P}^B = \mathrm{NP}^B$, then in fact we could not even settle the $\mathrm{P} = \mathrm{NP}$ question in either direction using relativizable techniques. In fact, Baker, Gill, and Solovay [BGS75] have created such oracles to draw these conclusions.

Most proofs in complexity theory relativize. For more details on the role of relativization, see [For94].

Consider the PSPACE-complete language TQBF. Note that

$$\mathrm{PSPACE}^{\mathrm{TQBF}} = \mathrm{PSPACE}.$$

By setting $A = \mathrm{TQBF}$, we have

Theorem 5.2 *There exists an oracle A such that $\mathrm{P}^A = \mathrm{PSPACE}^A$.*

Thus no relativizable proof exists that separates P from PSPACE.

If $\mathrm{P} = \mathrm{PSPACE}$ then all of the counting classes are equal to P and all are closed under full Turing reductions. Thus any proof that two counting classes differ or some counting class is not closed under some simple reduction would require nonrelativizing techniques.

Thus the only theorems that we can prove with current techniques will collapse classes and show closure properties. However, we can also use relativization to show the limits of this process.

Beigel [Bei94] has developed one of the more useful oracles pertaining to counting complexity.

Theorem 5.3 *There exists an oracle A such that $\mathrm{P}^{\mathrm{NP}^A}$ is not contained in PP^A.*

This oracle implies some important corollaries about the limits of relativizable techniques in counting complexity, some of which are listed below. These corollaries were known prior to Beigel's construction, but Beigel's work unified these results.

Corollary 5.4 *There exists a relativized world where*

1. PP *is not closed under Turing reductions.*
2. PP *does not contain the polynomial-time hierarchy.*
3. PP *does not contain* MP.

Proof:

1. If PP is closed under Turing reductions for all relativized worlds, then $\mathrm{P}^{\mathrm{NP}} \subseteq \mathrm{P}^{\mathrm{PP}} \subseteq \mathrm{PP}$, contradicting Theorem 5.3.
2. Immediate since $\mathrm{P}^{\mathrm{NP}} \subseteq \mathrm{PH}$.
3. Immediate since $\mathrm{PH} \subseteq \mathrm{MP}$. □

Yao's [Yao85, Hås89] original oracle Y giving a relativized world where

$$\mathrm{PH}^Y \neq \mathrm{PSPACE}^Y$$

actually shows that

$$\oplus\mathrm{P}^Y \not\subseteq \mathrm{PH}^Y.$$

Oracles also help us examine the relationships between NP, $\oplus$P, $\mathrm{C}_=\mathrm{P}$, and PP.

Theorem 5.5 ([Tor88, Bei91]) *There exists an oracle A such that*

1. $\oplus\mathrm{P}^A \not\subseteq \mathrm{PP}^A$.
2. $\mathrm{NP}^A \not\subseteq \oplus\mathrm{P}^A$ *(thus* $\mathrm{PP}^A \not\subseteq \oplus\mathrm{P}^A$*).*
3. $\mathrm{NP}^A \not\subseteq \mathrm{C}_=\mathrm{P}^A$ *(thus* $\mathrm{PP}^A \not\subseteq \mathrm{C}_=\mathrm{P}^A$*).*

It appears hard to guess at the power of SPP. In one sense, because of its restrictive definition it seems as though it should not have much power. However, the proof of Lemma 4.20 seems to indicate that SPP-like behavior may have tremendous power. Different oracles bring out this division:

Theorem 5.6 *There exist oracles A and B such that*

1. *The polynomial-time hierarchy is infinite relative to A, and* $\mathrm{P}^A = \mathrm{SPP}^A$.
2. *The polynomial-time hierarchy is infinite relative to B, and* SPP^B *strictly contains* PH^B.

One nagging relativization question remains open.

Open Question 5.7 *Does there exist an oracle A such that*

$$\mathrm{P}^{\#\mathrm{P}^A} \neq \mathrm{PSPACE}^A ?$$

In fact, even the following weaker question remains open.

Open Question 5.8 *Does there exist an oracle A such that*

$$\mathrm{PP}^{\oplus\mathrm{P}^A} \neq \mathrm{PSPACE}^A ?$$

6 Other Work

Such a short survey cannot possibly do justice to the large and vibrant area of counting complexity. Instead of taking on the near-impossible task of surveying the entire area, this paper has concentrated on looking at a few results in depth to give the reader some flavor of the importance of counting complexity. In this section, we would like to describe briefly some other work in the area and some of the applications to other areas.

6.1 Circuits

The techniques from counting complexity have played an important role in helping us understand the power of bounded-depth circuits. For a background in circuit complexity, we recommend the survey of Boppana and Sipser [BS90].

A PP question measures whether a certain number of computation paths accept. One can draw an analogy to threshold gates in a circuit that determine whether some number of inputs are true. Many of the techniques described in this survey have direct applications to circuit complexity, particularly in showing how to use threshold gates to simulate certain AND-OR circuits and also to use some number of threshold gates to simulate many of them.

For example, Allender [All89], Beigel, Reingold, and Spielman [BRS91] and Tarui [Tar93] show how to use Theorem 3.14 to simulate bounded-depth AND-OR circuits by small depth-2 circuits with a single threshold gate on top.

One can similarly draw an analogy between $\mathrm{Mod}_k\mathrm{P}$ computations and the class ACC of bounded depth circuits with Mod_k gates for some fixed k. Results of Yao [Yao90], Beigel and Tarui [BT94] and Green, Köbler, Regan, Schwentick, and Torán [GKR$^+$95] show how to simulate any ACC circuit by a depth-2 circuit with a symmetric gate on top. Allender and Gore [AG94] use these characterizations to separate uniform ACC from the counting class PP. Barrington [Bar92] gives a nice survey of these techniques and results.

6.2 Lowness

Complexity theorists often ask about the lowness of a class, i.e., the amount of information absorbed by a complexity class. Formally, for a class $\mathcal{C}$ we define $\mathrm{Low}(\mathcal{C})$ by

$$\mathrm{Low}(\mathcal{C}) = \{L \mid \mathcal{C}^L = \mathcal{C}\}.$$

Note that lowness depends not only on the membership of a class $\mathcal{C}$ but may also depend on the definition of $\mathcal{C}$, as well as the oracle access mechanism.

Some results on lowness:

Theorem 6.1

1. $\mathrm{Low}(\mathrm{NP}) = \mathrm{NP} \cap \text{co-NP}$.
2. $\mathrm{Low}(\mathrm{GapP}) = \mathrm{Low}(\mathrm{SPP}) = \mathrm{SPP}$ [FFK94].
3. $\mathrm{Low}(\mathrm{Mod}_k\mathrm{P}) = \mathrm{Mod}_k\mathrm{P}$ *for prime* k [PZ83, BG92, Her90].
4. $\mathrm{Low}(\mathrm{MP})$ *contains* PH *and* $\mathrm{Mod}_k\mathrm{P}$ *for all* k [GKR$^+$95].
5. $\mathrm{Low}(\mathrm{PSPACE}) = \mathrm{PSPACE}$.

A nice characterization of $\mathrm{Low}(\mathrm{PP})$ remains an interesting open question.

6.3 Characterizing Specific Problems

As mentioned in the introduction, Valiant [Val79a] defined the class #P specifically to capture the complexity of the permanent function. A permanent of a matrix is similar to the determinant except that all of the terms are added instead of subtracting alternating terms. While the determinant is easy to compute using Gaussian elimination, computing the permanent is #P-complete, i.e., $\#\mathrm{P} \subseteq \mathrm{FP}^{\mathrm{Permanent}}$.

In a later paper, Valiant [Val79b] showed the #P-completeness of several other combinatorial problems. Most counting versions of NP-complete problems are #P-complete, as are also some counting versions of polynomial-time computable problems like counting the number of simple paths or perfect matchings in a graph.

Köbler, Schöning, and J. Torán [KST92] used counting complexity to classify the graph isomorphism and automorphism problem. They show that SPP contains graph automorphism and Low(PP) contains graph isomorphism.

6.4 Interactive Proof Systems

Interactive proof systems developed by Babai [Bab85, BM88] and Goldwasser, Micali, and Rackoff [GMR89] give a probabilistic generalization of nondeterminism. In order to show a protocol for the coNP languages, Lund, Fortnow, Karloff, and Nisan [LFKN92] created an interactive proof system for the #P-complete permanent function. To this day, the only known protocols for coNP require counting complexity.

The Lund, Fortnow, Karloff, and Nisan protocol forms the core of much of the subsequent work in interactive proof systems that led to exciting applications in program checking and hardness results for approximation algorithms.

6.5 Counting in Space Classes

Instead of using nondeterministic polynomial-time Turing machines, one can also consider counting the accepting computations of nondeterministic logarithmic-space bounded machines. One can then define various classes at this level such as #L, GapL, PL, and $\oplus$L. One can show that the determinant function is complete for #L much the way that the permanent function is complete for #P [AO94].

Many of the results mentioned in this survey, such as Theorem 4.24, carry over to the log-space world.

6.6 Other Research

We still have not mentioned many other active areas of research in counting complexity. A small sampling includes the following.

- A counting hierarchy exists that is analogous to the polynomial-time hierarchy [Wag86].
- A general definition of "counting classes" [FFK94].
- The complexity of approximating counting functions [JVV86].
- Showing that closure properties of counting classes and functions imply various unknown collapses of other classes [OH91].
- Generalizations of counting to other functions on the accept and reject paths of nondeterministic machines [HLS$^+$93].

- Recent work in quantum complexity seems to deal with counting complexity except that counting is done in quite a different way. (See the survey by Berthiaume [Ber97] in this volume.)

Counting complexity has played an important role in theoretical computer science in the past two decades. We expect that the results and techniques we learned from this study will play a significant role in future research.

Acknowledgments: Thanks to Steve Homer for his careful reading of this paper. Thanks also to Fred Green for his help on the relationship between counting complexity and circuit complexity.

References

[AG94] E. Allender and V. Gore. A uniform circuit lower bound for the permanent. *SIAM Journal on Computing*, 23:1026–1049, 1994.

[All89] E. Allender. A note on the power of threshold circuits. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, pages 580–584. IEEE, New York, 1989.

[AO94] E. Allender and M. Ogihara. Relationships among PL, #L, and the determinant. In *Proceedings of the 9th IEEE Structure in Complexity Theory Conference*, pages 267–278. IEEE, New York, 1994.

[Bab85] L. Babai. Trading group theory for randomness. In *Proceedings of the 17th ACM Symposium on the Theory of Computing*, pages 421–429. ACM, New York, 1985.

[Bar92] D. Barrington. Quasipolynomial size circuit classes. In *Proceedings of the 7th IEEE Structure in Complexity Theory Conference*, pages 86–93. IEEE, New York, 1992.

[Bei91] R. Beigel. Relativized counting classes: relations among thresholds, parity and mods. *Journal of Computer and System Sciences*, 42(1):76–96, 1991.

[Bei94] R. Beigel. Perceptrons, PP and the polynomial hierarchy. *Computational Complexity*, 4:314–324, 1994.

[Ber97] A. Berthiaume. Quantum computation. In *Complexity Theory Retrospective II* (L. Hemaspaandra and A. Selman, eds.), chapter 2, this volume.

[BF91] L. Babai and L. Fortnow. Arithmetization: A new method in structural complexity theory. *Computational Complexity*, 1(1):41–66, 1991.

[BF95] H. Buhrman and L. Fortnow. Distinguishing complexity and symmetry of information. Technical Report TR 95-11, University of Chicago Department of Computer Science, 1995.

[BG92] R. Beigel and J. Gill. Counting classes: Thresholds, parity, mods, and fewness. *Theoretical Computer Science*, 103:3–23, 1992.

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

[BM88] L. Babai and S. Moran. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2):254–276, 1988.

[BRS91] R. Beigel, N. Reingold, and D. Spielman. The perceptron strikes back. In *Proceedings of the 6th IEEE Structure in Complexity Theory Conference*, pages 286–291. IEEE, New York, 1991.

[BRS95] R. Beigel, N. Reingold, and D. Spielman. PP is closed under intersection. *Journal of Computer and System Sciences*, 50(2):191–202, 1995.

[BS90] R. Boppana and M. Sipser. The complexity of finite functions. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 14, pages 757–804. North-Holland, Amsterdam, 1990.

[BT94] R. Beigel and J. Tarui. On ACC. *Computational Complexity*, 4:350–366, 1994.

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

[For94] L. Fortnow. The role of relativization in complexity theory. *Bulletin of the European Association for Theoretical Computer Science*, 52:229–244, February 1994.

[FR96] L. Fortnow and N. Reingold. PP is closed under truth-table reductions. *Information and Computation*, 124(1):1–6, 1996.

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

[GKR+95] F. Green, J. Köbler, K. Regan, T. Schwentick, and J. Torán. The power of the middle bit of a #P function. *Journal of Computer and System Sciences*, 50(3):456–467, 1995.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[Hås89] J. Håstad. Almost optimal lower bounds for small depth circuits. In S. Micali, editor, *Randomness and Computation*, volume 5 of *Advances in Computing Research*, pages 143–170. JAI Press, Greenwich, 1989.

[Her90] U. Hertrampf. Relations among MOD classes. *Theoretical Computer Science*, 74:325–328, 1990.

[HLS+93] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, and K. Wagner. On the power of polynomial time bit-reductions. In *Proceedings of the 8th IEEE Structure in Complexity Theory Conference*, pages 200–207. IEEE, New York, 1993.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, Mass., 1979.

[HV95] L. Hemaspaandra and H. Vollmer. The satanic notations: Counting classes beyond #P and other definitional adventures. *SIGACT News*, 26(1):2–13, 1995.

[JVV86] M. Jerrum, L. Valiant, and V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986.

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

[LFKN92] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992.

[MS72] A. Meyer and L. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of the 13th IEEE Symposium on Switching and Automata Theory*, pages 125–129. IEEE, New York, 1972.

[MVV87] K. Mulmuley, U. Vazirani, and V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[New64] D. Newman. Rational approximations to $|x|$. *Michigan Mathematics Journal*, 11:11–14, 1964.

[Nis91] N. Nisan. Pseudorandom bits for constant-depth circuits. *Combinatorica*, 11(1):63–70, 1991.

[NW94] N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[OH91] M. Ogiwara and L. Hemachandra. A complexity theory of feasible closure properties. In *Proceedings of the 6th IEEE Structure in Complexity Theory Conference*, pages 16–29. IEEE, New York, 1991.

[PZ83] C. Papadimitriou and S. Zachos. Two remarks on the power of counting. In *Proceedings of the 6th GI Conference on Theoretical Computer Science*, pages 269–276. Volume 145, Lecture Notes in Computer Science, Springer, Berlin, 1983.

[Rog87] H. Rogers. *Theory of Recursive Functions and Effective Computability*. MIT Press, Cambridge, 1987.

[Sch90] U. Schöning. The power of counting. In A. Selman, editor, *Complexity Theory Retrospective*, pages 204–223. Springer, New York, 1990.

[Sho90] V. Shoup. New algorithms for finding irreducible polynomials over finite fields. *Mathematics of Computation*, 54:435–447, 1990.

[Tar93] J. Tarui. Probabilistic polynomials, AC^0 functions and the polynomial-time hierarchy. *Theoretical Computer Science*, 113:167–183, 1993.

[TO92] S. Toda and M. Ogiwara. Counting classes are at least as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 21(2):316–328, 1992.

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

[Tor88] J. Torán. Structural properties of the counting hierarchies. Ph.D. thesis, Facultat d'Informàtica, UPC Barcelona, January 1988.

[Val79a] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[Val79b] L. Valiant. The complexity of reliability and enumeration problems. *SIAM Journal on Computing*, 8:410–421, 1979.

[VV86] L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

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

[Yao85] A. Yao. Separating the polynomial-time hierarchy by oracles. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, pages 1–10. IEEE, New York, 1985.

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