Locally Random Reductions in Interactive Complexity Theory

Joan Feigenbaum
AT&T Bell Labs, Rm 2C-473
600 Mountain Avenue
Murray Hill, NJ 07974-0636 USA
jf@research.att.com

Abstract

We survey definitions, known results, and open questions in the area of locally random reductions and explore the ramifications of these reductions in complexity theory.

This paper is based on a survey talk given at the DIMACS Workshop on Structural Complexity and Cryptography, Rutgers University, New Brunswick NJ, December 1990.

1 Introduction

We consider the question of whether a probabilistic polynomial-time machine A can compute a function f in the following manner. A interacts with one or more machines B_1, ..., B_k that are not restricted to probabilistic polynomial time. At the end of the interaction, A can use the information obtained from the B_i's to compute $f(x)$. However, the information that A sends to the B_i's is *locally random*. Informally, this means that no individual B_i can use it to figure out what A's private input x is.

This study can be motivated by the practical problem of using shared resources for private computations. For example, f may be a financial service provided, for a fee, by the B_i's, and x may be A's personal financial position. Alternatively, the B_i's could be publicly supported supercomputing facilities, and A could be one of many competing scientists trying to use them without revealing sensitive data.

The focus of this paper, however, is on another motive for studying locally random reductions, namely their importance in complexity theory. For instance, they provide a way of relating the worst-case complexity of one function to the average-case complexity of another. Suppose that each of the B_i's computes the function g. On input x, the machine A produces random instances in $Dom(g)$ according to some polynomial-time sampleable distribution D and feeds them to the B_i's. From the B_i's answers to these random instances, A can compute $f(x)$ in polynomial time. Thus, if there is an algorithm for g that is fast-on-average, with respect to the distribution D, there is a randomized algorithm for f that is fast in the worst case. If $f = g$, then f is hard on average, with respect to D, if it is hard in the worst case. More generally, locally random reductions play an important role in what I refer to informally as *interactive complexity theory.* By this, I mean complexity-theoretic questions about models of computation involving interaction between two or more computers, in which these computers (usually) have different amounts of resources; topics in this theory include interactive proof systems, program checking, and program self-testing and -correcting.

Among the issues we must answer in formalizing these notions are the following.

1. What does it mean to say that A's input x is not revealed to the B_i's? Suppose that some subset of the B_i's collude – how big do the coalitions have to be in order

to have enough information to recover x?

2. How many random queries must A ask? In particular, how big does k have to be if x is of size n?

3. How many rounds of interaction must A have with the B_i's? Are there functions f for which A can ask all of the random questions at once, or must A always follow an *adaptive* strategy, basing later questions on answers to earlier questions?

4. How powerful must the B_i's be? For example, if each B_i computes the function g, what is the relationship of the complexity of g to that of f? When can we take $g = f$?

5. Suppose that there are multiple rounds of interaction. Does B_i's answer in round q have to be based only on A's question in round q, or can it be based on the questions in rounds 1 through q? That is, when should B_i also follow an adaptive strategy?

This paper surveys definitions, known results, and open questions in the area of locally random reductions and explores the ramifications of these reductions in complexity theory. One of the main purposes of this paper is to establish a unified notation and terminology with which to state what is known; therefore, some of the results from the literature are stated here in different terms from those used by the original authors.

We end this introduction with two standard examples. These examples arise in the theory of cryptography, e.g., in cryptographic protocols for *probabilistic encryption* [GM], *pseudorandom number generation* [BlMi], and *zero-knowledge proof systems* [GMR].

Example 1.1 *The Discrete Logarithm Function:*

Let p be a prime, g a generator of the cyclic group $\mathcal{Z}_p^*$, and u an element of $\mathcal{Z}_p^*$. The discrete logarithm function f maps the triple (u, g, p) to the unique $e \in [0, p-1]$ such that $u \equiv g^e \bmod p$. How could a probabilistic polynomial-time machine A obtain $f(u, g, p)$ without revealing u? First, A chooses an exponent c uniformly at random from $[0, p-1]$. A then computes $v \equiv u \cdot g^c \bmod p$ and obtains $e' = f(v, g, p)$. The discrete logarithm e of (u, g, p) is just $e' - c \bmod p - 1$. Because c is uniformly distributed over $[0, p-1]$, v is uniformly distributed over $\mathcal{Z}_p^*$, and the random instance (v, g, p) reveals no information about u.

Example 1.2 *The Quadratic Residuosity Function:*

Let m be the product of primes p and q that are both congruent to 3 mod 4. Let u be an element of $\mathcal{Z}_m^*[+1]$, the elements of $[1, m-1]$ that are relatively prime to m and have Jacobi symbol $+1$. The quadratic residuosity function f maps (u, m) to 1 if there is an element a of $\mathcal{Z}_m^*$ such that $a^2 \equiv u \bmod m$, and f maps (u, m) to 0 otherwise. Because m is the product of two primes that are congruent to 3 mod 4, $f(-1, m) = 0$. A probabilistic polynomial-time machine A could obtain the quadratic residuosity e of u mod m without revealing u as follows. A first chooses b uniformly at random from $\{0, 1\}$ and d uniformly at random from $\mathcal{Z}_m^*$. It then computes $v = u \cdot d^2 \cdot (-1)^b \bmod m$. A obtains the value $e' \equiv f(v, m)$; if $b = 0$, then $e = e'$, and if $b = 1$, then $e = 1 - e'$. The element v is distributed uniformly at random over $\mathcal{Z}_m^*[+1]$ and thus reveals no information about u.

For number-theoretic background and a more detailed explanation of these examples, see [AFK, AL].

2 Terminology and Notation

Each function considered in this paper is defined on some infinite domain I. There is a sequence of subdomains $I_1, I_2, \ldots$, such that $I = \cup_{n \geq 1} I_n$ and a constant c such that, for all $x \in I_n$, $n^{1/c} \leq |x| \leq n^c$, where $|x|$ is the number of bits in some suitable binary encoding of x. For example, we may have $I = \{0, 1\}^+$ and $I_n = \{0, 1\}^n$, in which case we can take $c = 1$. If p_n is an n-bit prime, then I_n may consist of n-tuples of elements in GF(p_n), in which case we can take $c = 2$. We refer to elements of I_n as "inputs of size n." So "size" is a generalization of binary length.

By a *probabilistic polynomial-time machine* M, we mean a machine that has access to a source of unbiased random bits and always halts in a fixed polynomial number of steps, but may output a wrong value. (Thus we do not mean a machine that always outputs the correct value but can only be guaranteed to halt in expected polynomial time.) If M is a probabilistic polynomial-time oracle machine, we use the notation $M^f(x)$ for the output of M when it is run with input x and an oracle for the function f. Note that $M^f(x)$ is a random variable. If Q is a program that purports to compute f, then the notation $M^Q(x)$ has the natural meaning; if Q is a probabilistic program, then the output distribution is determined by the algorithms and the coin-toss sequences of both M and Q.

The probabilistic polynomial-time oracle machine M is a *downward self-reduction for* f if, for all n and all $x \in I_n$, $M^f(x) = f(x)$ with probability at least 3/4, and all oracle queries that M makes on input x are of size less than n.

A set $S = (A, B_1, \ldots, B_k)$ of *private-input, interactive Turing machines* is defined as follows. The input x, which is of size n, is given only to A – that is the sense in which the set of machines is "private." The parameter $k(n)$ is a polynomially bounded function of n; thus, there are really an infinite number of B_i's, but A only interacts with $k(n)$ of them on inputs of size n. A is a probabilistic polynomial-time machine. Each B_i consists of an arbitrary oracle O_i and a probabilistic oracle Turing machine M_i computing $M_i^{O_i}$; for each x, $M_i^{O_i}(x)$ must be bounded in size by a polynomial in n. (Alternatively, B_i can be thought of as an arbitrary distribution on functions that map inputs of size n to outputs of size bounded by a polynomial in n; Fortnow and Kurtz have shown the equivalence of these two definitions [FK].) For each i, there is a private communication channel between A and B_i; so B_j has no access to the messages exchanged by A and B_i if $j \neq i$. In *round* q of an execution of $(A, B_1, \ldots, B_k)$ on input x, A does a polynomial-time local computation and sends a message a_{iq} to each B_i, and each B_i does an arbitrary local computation and sends a message b_{iq} to A. The messages $\{a_{iq}\}$ are based on x, $\{b_{js}\}$, for $1 \leq j \leq k$ and $1 \leq s < q$, and A's random coin-tosses. The message b_{iq} is based on B_i's random coin-tosses and on $\{a_{is}\}$, for $1 \leq s \leq q$. The number of rounds of communication and the maximum size of a message are both polynomially bounded in n. After the final round of communication, A does a last polynomial-time computation and outputs a value, which is denoted $(A(x), B_1, \ldots, B_k)$. Note that $(A(x), B_1, \ldots, B_k)$ is a random variable.

$S = (A, B_1, \ldots, B_k)$ is a set of *shared-input, interactive Turing machines* if it has all of the above properties, with this exception: The input x is given to all of the B_i's as well as to A. In this case, B_i may of course use x in the computation of a message b_{iq}. A's output in this case is denoted $(A, B_1, \ldots, B_k)(x)$.

For any subset $\{i_1, \ldots, i_t\}$ of $\{1, \ldots, k\}$, $Trans_S(x, i_1, \ldots, i_t)$ is a random variable consisting of the messages exchanged by A and B_{i_1}, ..., B_{i_t} on input x. ($Trans$ is for "transcript.")

The probabilistic polynomial-time machine A is sometimes called the *querier* and B_1, ..., B_k the *powerful players.* If B_i is deterministic and, in each round q, bases his answer b_{iq} solely on A's query current a_{iq}, then B_i is called *functional* (or *memoryless*). In the more general case, when B_i may also use coin-tosses and A's queries from earlier rounds, B_i is called *adaptive.*

3 Basic Definitions

Throughout this section x is the input to a randomized reduction, n is the size of x, and ran is a string chosen uniformly at random from $\{0,1\}^{w(n)}$, where w is some polynomially bounded function of n. The parameters k (for the number of powerful players) and r (for the number of rounds) are also polynomially bounded functions of n.

Definition 3.1 *A* **nonadaptive $k(n)$-locally random reduction** *(abbreviated "k-lrr")* **from f to $(g1, \ldots, gk)$** *is a pair of polynomial-time computable functions σ and ϕ with the following properties.*

(1) For all n and all $x \in I_n$,

$$f(x) = \phi(x, ran, g1(\sigma(1, x, ran)), \ldots, gk(\sigma(k, x, ran))),$$

for at least $3/4$ of all ran's in $\{0,1\}^{w(n)}$, and

(2) For all x_1 and x_2 of size n and all $i \in \{1, \ldots, k\}$, if ran is chosen uniformly at random, then $\sigma(i, x_1, ran)$ and $\sigma(i, x_2, ran)$ are identically distributed.

Note that there is no bound on the complexity of gi with respect to the complexity of f. It is actually straightforward to show that there is no loss of generality in requiring that $g1 = \cdots = gk$, but it is convenient in some proofs to allow the gi's to be different.

Definition 3.2 *A* **nonadaptive $k(n)$-random-self-reduction** *(abbreviated "k-rsr")* **for f** *is a nonadaptive k-lrr from f to $(f, \ldots, f)$.*

The word *nonadaptive* in Definitions 3.1 and 3.2 refers to the fact that the querier produces all of the random queries in one round; that is, $\sigma(i, x, ran)$ does not depend on $gj(\sigma(j, x, ran))$ for $j \neq i$. The next two definitions cover the case in which the querier A follows an adaptive strategy, but the powerful players B_1, ..., B_k are functional.

Definition 3.3 *An* $\boldsymbol{r(n)}$**-round, adaptive** $\boldsymbol{k(n)}$**-locally random reduction from** $\boldsymbol{f}$ **to** $(\boldsymbol{g_1, \ldots, g_k})$ *is a set S of private-input, interactive Turing machines $(A, B_1, \ldots, B_k)$ with the following properties.*

(1) For all i, $1 \leq i \leq k$, the i^{th} powerful player B_i is functional and computes the function gi.

(2) For all n, the number of rounds of interaction on inputs of size n is at most $r(n)$.

(3) For all n and all $x \in I_n$,

$$\mathrm{Prob}((A(x), B_1, \ldots, B_k) = f(x)) \geq 3/4.$$

(4) For all x_1 and x_2 of size n and all $i \in \{1, \ldots, k\}$, the random variables $Trans_S(x_1, i)$ and $Trans_S(x_2, i)$ are identically distributed.

Definition 3.4 *An* $\boldsymbol{r(n)}$**-round, adaptive** $\boldsymbol{k(n)}$**-random-self-reduction for** $\boldsymbol{f}$ *is an $r(n)$-round k-lrr from f to $(f, \ldots, f)$.*

Finally, we consider a type of reduction in which both the querier A and the powerful players $\{B_i\}$ are allowed to use randomized, adaptive strategies.

Definition 3.5 *An* $\boldsymbol{r(n)}$**-round,** $\boldsymbol{k(n)}$**-instance-hiding scheme** *(abbreviated "k-ihs")* **for** $\boldsymbol{f}$ *is a set $(A, B_1, \ldots, B_k)$ of private-input, interactive Turing machines that satisfy properties (2) through (4) of Definition 3.3 but do not necessarily satisfy property (1). That is, in an instance-hiding scheme, the powerful players B_i may be adaptive.*

Property (2) in Definitions 3.1 and 3.2 and property (4) in Definitions 3.3, 3.4, and 3.5 are referred to as *the instance-hiding property.* This means that the querier is successful in hiding everything about the instance x, except its size, from each individual powerful player. It can also be called the *local randomness property*: From the local viewpoint of each powerful player, the communication received from A looks random; from the global viewpoint (i.e., that of someone who sees all of the messages received by all of the players), the communication may reveal x. Note that the instance-hiding property is not required to hold if the powerful players cheat. That is, queries in later rounds may reveal more than the size if the querier constructs them using wrong answers from earlier rounds.

There are two more parameters that are often added to these definitions in order to make them more general: the *privacy parameter* $t(n)$, which is a polynomially bounded function of n, and the *leak function* L, which is defined on $Dom(f)$. A reduction is *t-private* if the privacy of the input is preserved even if as many as t of the powerful players collude. Formally, the instance-hiding property must be restated as follows: For all x_1 and x_2 of size n, for all $\{i_1, \ldots, i_t\} \subseteq \{1, \ldots, k\}$, the random variables $Trans_S(x_1, i_1, \ldots, i_t)$ and $Trans_S(x_2, i_1, \ldots, i_t)$ are identically distributed. Intuitively, the leak function L denotes some piece of information about the input x that may be revealed to the powerful players even if they don't collude. That is, the querier may have to leak *some* information about x if the output of the reduction is to be meaningful at all; in most examples, this information is the size of x, but the notion is perfectly well-defined for other leak functions. Formally, a t-private reduction *leaks at most* L if, for all x_1 and x_2 such that $L(x_1) = L(x_2)$, for all $\{i_1, \ldots, i_t\} \subseteq \{1, \ldots, k\}$, the random variables $Trans_S(x_1, i_1, \ldots, i_t)$ and $Trans_S(x_2, i_1, \ldots, i_t)$ are identically distributed.

Defaults:

1. Unless otherwise specified, the leak function $L(x)$ is assumed to be n, the size of x.
2. Unless otherwise specified, the privacy parameter $t(n)$ is assumed to be 1. That is, each powerful player learns at most $L(x)$ in isolation, but if 2 or more of them collude, they may learn x.
3. Unless specified, both k and r are assumed to be arbitrary polynomials.
4. If only one target function g is specified in a locally random reduction, it means that $g1 = \cdots = gk = g$.
5. Unless specified, a reduction is allowed to be adaptive.

For example, an "rsr for f" is a 1-private, $poly(n)$-round, adaptive $poly(n)$-rsr for f that leaks at most n; a "nonadaptive k-lrr from f to g" is a 1-private, nonadaptive k-lrr from f to $(g, \ldots, g)$ that leaks at most n.

Example 1.1 shows that the discrete logarithm function is nonadaptively 1-rsr, leaking at most g and p. Example 1.2 shows that the quadratic residuosity function is nonadaptively 1-rsr, leaking at most m.

Question 3.6 *Is there a nontrivial leak function L such that integer factoring is rsr leaking at most L?*

In all of the reductions defined in this section, as well as some that we define later, the upper bound on the error probability is given as 1/4. It can always be made exponentially small by increasing the number of queries by a suitable polynomial factor. This is done by running independent copies of the reduction and taking the plurality of the answers to be the output. The proof that the error bound goes down exponentially is a standard argument using Chernoff bounds.

A reduction is called *size-preserving* if, for all inputs $x \in I_n$, all of the random queries made are of size at most n.

Instance-hiding schemes were first studied in [AFK]; there, they are called "encryption schemes for functions," and only schemes with one powerful player B are considered. The term "instance-hiding scheme" first appeared in [BeaF], which is the first paper to prove anything about schemes with multiple powerful players. The term "locally random reduction" first appeared in [BFKR], where it refers to what we here call a "nonadaptive locally random reduction."

Random-self-reducibility and its effect on the relationship of average-case complexity to worst-case complexity was used in the design and analysis of cryptographic protocols at least as early as [BlMi, GM]. It was first defined formally and studied in its own right from a complexity theoretic point of view in [AFK]. The definitions of rsr that we use here are almost identical to those in [FF].

4 Low-Degree Polynomials

Let $f : \{0,1\}^* \rightarrow \{0,1\}$ be an arbitrary boolean function, and let $f_n : \{0,1\}^n \rightarrow \{0,1\}$ be the restriction of f to inputs $(x_1, \ldots, x_n) \in \{0,1\}^n$. For any finite field K_n, there is a unique multilinear polynomial $g_n \in K_n[X_1, \ldots, X_n]$ that *represents* f_n over K_n – i.e., g_n agrees with f_n on all inputs $(x_1, \ldots, x_n)$ in $\{0,1\}^n$. In this paper, K_n is always a field of size at least $n+1$. The polynomial g_n has a standard explicit formula with some important computational properties. Let $x = (x_1, \ldots, x_n)$ be an arbitrary element of K_n^n and $y = (y_1, \ldots, y_n)$ be an arbitrary element of $\{0,1\}^n$.

$$g_n(x) = \sum_{y \in \{0,1\}^n} \delta_y(x) f_n(y) \tag{1}$$

$$\delta_y(x) = \prod_{i=1}^{n} (x_i - (1 - y_i))(-1)^{(1-y_i)} \tag{2}$$

For $x \in \{0,1\}^n$, the monomial $\delta_y(x)$ is 1 if $y = x$, and it is 0 otherwise. We call g_n the *arithmetization of* f_n *over* K_n and $g = \{g_n\}_{n\geq 1}$ the arithmetization of f over $\{K_n\}_{n\geq 1}$.

The following basic result, due to Beaver and Feigenbaum, is the starting point for most of the positive results discussed in this paper.

Theorem 4.1 [BeaF]: *Let* $f = \{f_n\}_{n\geq 1}$ *be an arbitrary boolean function,* $\{K_n\}_{n\geq 1}$ *be a* *sequence of finite fields such that $|K_n| > n$, and $g = \{g_n\}_{n\geq 1}$ be the arithmetization of f over $\{K_n\}_{n\geq 1}$. Then f is nonadaptively $(n+1)$-lrr to g.*

Proof: Recall that $g_n \in K_n[X_1, \ldots, X_n]$ is of total degree at most n. The reduction proceeds as follows. Let α_1, …, α_{n+1} be distinct elements of K_n. Choose coefficients c_1, …, c_n independently and uniformly at random from K_n. Let $\sigma(i, (x_1, \ldots, x_n), ran) = (c_1\alpha_i + x_1, \ldots, c_n\alpha_i + x_n)$ for $1 \leq i \leq n+1$.

Consider the one-variable polynomial G defined by

$$G(Z) = g_n(c_1 Z + x_1, \ldots, c_n Z + x_n).$$

Note that G is of degree at most n and hence determined by its values on $n+1$ distinct values of Z. If $y_i = \sigma(i, (x_1, \ldots, x_n), ran)$, then G satisfies

$$G(\alpha_i) = g_n(y_i).$$

It also satisfies

$$G(0) = g_n(x_1, \ldots, x_n).$$

The function ϕ of the lrr interpolates the $n+1$ values $(\alpha_1, G(\alpha_1)), \ldots, (\alpha_{n+1}, G(\alpha_{n+1}))$ to recover the polynomial G and outputs the constant term. ∎

Beaver and Feigenbaum stated this theorem using the term "instance-hiding scheme." The term "locally random reduction" was later introduced in [BFKR], where it was pointed out that the construction in [BeaF] was actually a locally random reduction.

Lipton subsequently observed that the Beaver-Feigenbaum construction also shows that low-degree multivariate polynomials are themselves nonadaptively rsr. He also pointed out that the permanent function for matrices over finite fields is a low-degree polynomial. This has implications for the complexity class #P, as we shall see in Section 6. Lipton uses the term "randomly testable" for what we here refer to as "random-self-reducible."

Theorem 4.2 [Li]: *If $g_n \in K_n[X_1, \ldots, X_n]$ has total degree d_n and $|K_n| > d_n$, where d_n is bounded by a polynomial in n, then $g = \{g_n\}_{n\geq 1}$ is nonadaptively (d_n+1)-rsr. In particular, if $K_n = \mathrm{GF}(p_n)$, where $p_n > n$, and g_n is the permanent of $n \times n$ matrices over K_n, then g_n is nonadaptively $(n+1)$-rsr.*

The following improvement of Theorem 4.1 is due to Beaver, Feigenbaum, Kilian, and Rogaway.

Theorem 4.3 [BFKR]: *Let $f = \{f_n\}_{n \geq 1}$ be a boolean function, $t(n)$ a polynomial, and $\{K_n\}_{n \geq 1}$ a sequence of finite fields such that $|K_n| > nt(n)/\log n$. Then there is a family $h = \{h_n\}_{n \geq 1}$ of multivariate polynomials over $\{K_n\}_{n \geq 1}$ such that f has a $t(n)$-private, nonadaptive $(1 + (nt(n)/\log n))$-lrr to h.*

Proof: The proof is divided into two lemmas.

Lemma 4.4 *Every boolean function $f = \{f_n\}_{n \geq 1}$ is polynomial-time, many-one reducible to a family $h = \{h_n\}_{n \geq 1}$ of multivariate polynomials over K_n, in which h_n is of total degree $n/\log n$.*

Lemma 4.5 *Let $t(n)$ and $d(n)$ be polynomially bounded functions and $\{K_n\}_{n \geq 1}$ be a sequence of finite fields with $|K_n| > t(n) \cdot d(n)$. Let h_n be a multivariate polynomial of total degree $d(n)$ over K_n. Then $h = \{h_n\}_{n \geq 1}$ is t-privately, nonadaptively $(1 + dt)$-rsr.*

Assume without loss of generality that $\log n$ divides n; if it does not, then x can be "padded" with dummy input bits. Divide the input bits $x_1, \ldots, x_n$ into consecutive "blocks" of size $\log n$. For example, if $n = 4$, then the first block is $\{x_1, x_2\}$ and the second is $\{x_3, x_4\}$.

Proof of Lemma 4.4: The crux of this construction is a change of variables that allows us to use a total-degree-$(n/\log n)$ polynomial h_n in $n^2/\log n$ variables in place of the total-degree-n polynomial g_n in n variables that was used in Theorem 4.1.

Let $\{x_j\}_{j=(l-1)\log n+1}^{l \log n}$, $1 \leq l \leq n/\log n$, be a block of input bits and $\{X_j\}_{j=(l-1)\log n+1}^{l \log n}$ be the corresponding block of indeterminates over K_n that are used to define the arithmetization g of f. For each subset S of the indices $\{(l-1)\log n + 1, \ldots, l \log n\}$, let the variable W_S represent the monomial $\Pi_{j \in S} X_j$. There are $n/\log n$ blocks and hence a total of $n^2/\log n$ variables $W_1, \ldots, W_{n^2/\log n}$.

Each monomial in g_n can be represented as a monomial in $n/\log n$ of the W's. Let $h_n(W_1, \ldots, W_{n^2/\log n})$ be the total-degree-$(n/\log n)$ polynomial that results from summing these representations of all of the monomials in g_n. ∎

Proof of Lemma 4.5: Let $h_n(W_1, \ldots, W_s)$ be a total-degree-$d(n)$ polynomial over K_n, and let $w = (w_1, \ldots, w_s)$ be an element of K_n^s. We show how to reduce w to a collection of $1+dt$ elements of K_n^s with an appropriate distribution. The main idea of this construction is to use a random degree-t univariate polynomial for each W_l instead of the random degree-1 univariate polynomial that was used in Theorem 4.1.

Let α_1, ..., α_{1+td} be distinct elements of K_n. For each $1 \leq l \leq s$, choose t elements $c_{l,1}$, ..., $c_{l,t}$ of K_n independently and uniformly at random, and let $q_l(Z) = c_{l,t}Z^t + \cdots + c_{l,1}Z + w_l$. For $1 \leq i \leq 1+td$, the function σ maps (i, x, ran) to $y_i = (q_1(\alpha_i), \ldots, q_s(\alpha_i))$. Then the univariate polynomial

$$H(Z) = h_n(q_1(Z), \ldots, q_s(Z))$$

is of degree at most td and satisfies

$$H(\alpha_i) = h_n(y_i)$$

and

$$H(0) = h_n(w_1, \ldots, w_s).$$

The function ϕ recovers $h_n(w)$ by interpolating the polynomial $H(Z)$ from the pairs $(\alpha_i, H(\alpha_i))$ and outputting the constant term.

This reduction has the instance-hiding property for the same reason that Shamir's secret-sharing scheme works [S1]: For every l, t or fewer values of the random degree-t polynomial q_l reveal no information about its constant term w_l. ∎

Remark 4.6 *This can be improved (in the sense that the parameter $k(n)$ can be reduced) to a $t(n)$-private, nonadaptive $(1 + (nt(n)/c\log n))$-lrr, for any positive constant c, by increasing the block size to $c \log n$ (and the number of x_j's per block to n^c).*

Remark 4.7 *More generally, one can use block size l, which is not necessarily $O(\log n)$. This will result in a $t(n)$-private, nonadaptive $(1 + t(n) \cdot \lceil n/l \rceil)$-lrr from f to polynomials h_n of total degree $\lceil n/l \rceil$ in $2^l \cdot \lceil n/l \rceil$ variables, but the reduction will take time $2^l \cdot poly(n)$.*

Question 4.8 *Can the construction of Theorem 4.3 be improved? For example, is it true that there is some function $k(n) = o(n/\log n)$ such that, for every boolean function*

f, there exists a function f' such that f is nonadaptively k-lrr to f'? Is there such a construction in which f' is a sequence of multivariate polynomials? Does adaptiveness help? That is, is there a way to use $o(n/\log n)$ powerful players if they or the querier or both may use an adaptive strategy?

5 Program Checkers and Self-Tester/Correctors

Throughout this section, Q is a program that purports to compute the function f.

Definition 5.1 [Bl]: *The probabilistic polynomial-time oracle machine Ch is a* **checker** *for f if it has the following properties. The output of Ch is always* CORRECT *or* FAULTY*. On input x, Ch first makes the oracle query $Q(x)$ to obtain Q's claimed value y for $f(x)$. On all inputs x, $Ch^f(x) =$* CORRECT *with probability at least 3/4. If $y \neq f(x)$, then $Ch^Q(x) =$* FAULTY *with probability at least 3/4.*

Note that the output of a checker may be CORRECT or FAULTY if $Q(x) = f(x)$ but $Q(x') \neq f(x')$ for some $x' \neq x$. We say that f is *checkable* if it has a checker. We say that a set $S \subseteq \{0,1\}^*$ is checkable if its characteristic function is checkable.

The following notion was first defined by Blum, Luby, and Rubinfeld. The definition presented here, due to Babai, Fortnow, and Lund, is slightly more general and is better suited to a complexity theoretic treatment than the original definition in [BLR1].

Definition 5.2 [BFL]: *The probabilistic polynomial-time oracle machines T and Co are a* **self-testing/correcting pair** *for f if they behave as follows. The output of T is always* PASS *or* FAIL*. For all n, $T^f(1^n) =$* PASS *with probability at least 3/4. If the probability that $T^Q(1^n) =$* PASS *is at least 1/4, then for any $x \in I_n$, the probability that $Co^Q(x) = f(x)$ is at least 3/4.*

Intuitively, the output of T, the tester, on input 1^n and oracle Q, is supposed to be PASS if Q is correct on I_n, and it's supposed to be FAIL if Q is incorrect on I_n. Formally, Definition 5.2 requires two things. The first is that, if Q is in fact correct everywhere (i.e., if $Q = f$), then T outputs PASS with high probability. On the other hand, T may still output PASS if Q is incorrect somewhere. However, in this case, the second formal requirement says that, if $T^Q(1^n)$ outputs PASS with any significant probability, then Co, the corrector, must be able to compensate for these errors in Q, i.e., $Co^Q(x)$ must be correct with high probability for any x in I_n.

The following observation was made originally by Blum, Luby, and Rubinfeld. We include it here in order to demonstrate that it holds for the definitions we are using here as well as for the ones in [BLR1].

Observation 5.3 *If f has a self-testing/correcting pair, then it has a checker.*

Proof: Let T and Co be a self-testing/correcting pair for f. The checker Ch proceeds as follows. It first queries the program Q to obtain $y = Q(x)$. If $x \in I_n$, Ch then runs $T^Q(1^n)$. If the output is FAIL, then Ch outputs FAULTY. If T outputs PASS, then Ch runs Co to obtain $y' = Co^Q(x)$. It outputs CORRECT if $y = y'$ and FAULTY if $y \neq y'$.

It is easy to see that Ch satisfies Definition 5.1. If $Q = f$, then T will output PASS with high probability and $Co^Q(x)$ will be equal to $f(x)$ (which is also equal to y) with high probability. If $y \neq f(x)$, then either T will output FAIL, or with high probability $Co^Q(x)$ will not be equal to y. ∎

The next theorem has important consequences in the theory of multiprover, interactive proof systems, as we shall see in Section 6. Once again, this theorem was originally proven by Blum, Luby, and Rubinfeld; we include a slight modification of their proof in order to show that it holds for the more general definitions used here as well as for the ones in [BLR1]. One significant thing to note is that Blum, Luby, and Rubinfeld assume that the random-self-reductions are nonadaptive and that k, the number of random queries, is constant. As the following theorem shows, neither of these assumptions is necessary.

Theorem 5.4 *If f has a downward self-reduction and a size-preserving rsr, then f has a self-testing/correcting pair.*

Sketch of proof: We construct a self-testing/correcting pair T, Co by induction on n. Let D be a downward self-reduction for f and R be the probabilistic polynomial-time machine in a size-preserving rsr for f. The program Q will be used by D and R in place of the function f.

If $n = 1$, then T and Co just compare the outputs of Q with values stored in a table. Assume that $n > 1$ and that T and Co have been correctly specified for inputs of size less than n.

On input 1^n and oracle Q, the tester T first chooses an instance $x \in I_n$. It then repeats the following procedure m times, where m is a sufficiently large polynomial in n. For each i, $1 \le i \le m$, it uses D and R, together with T and Co on inputs of size less than n, to compute a candidate y_i for $f(x)$. If T outputs FAIL on some input 1^j, $j < n$, during any of these computations, then it also outputs FAIL on the original input 1^n. Otherwise, it outputs PASS if and only if $y_i = Q(x)$ for all i.

The candidate y_i is computed as follows. Run the rsr R on input x. Suppose that z is an oracle query produced in this run. To compute $f(z)$, run D on z and use T, Co, and Q to answer all oracle queries. That is, if z' is an oracle call made by D, and $z' \in I_j$, first run $T^Q(1^j)$. If it outputs FAIL, then stop – the top-level call to the tester will be FAIL in this case. Otherwise, use $Co^Q(z')$ as the value $f(z')$. This will be correct with high probability, because $j < n$. If R runs to completion, then its output is y_i.

For any input x, the self-corrector Co simply outputs $R^Q(x)$. ∎

Numerous examples of functions that have checkers and self-testing/correcting pairs can be found in [Bl, Ka, Ru].

Question 5.5 *Find other natural properties of functions that imply the existence of checkers or self-testing/correcting pairs.*

6 Complexity-Theoretic Implications

In this section, we explore some of the complexity-theoretic consequences of the results discussed in Sections 4 and 5.

Theorem 6.1 *All #P-complete functions are rsr.*

Proof: This follows easily from Theorems 4.1 and 4.2. Let PERM be the $\#P$-complete function that computes permanents of integer-matrices. An instance x of PERM can be reduced to the computation of PERM(x) mod p_j, for some small collection of primes p_j – to recover PERM(x) from {PERM(x) mod p_j}, use the Chinese Remainder Theorem. For each p_j, PERM(x) mod p_j is just a low-degree polynomial over a finite field and is thus reducible to a polynomial-sized collection of random instances {PERM(y_{ij}) mod p_j}. These y_{ij}'s can be regarded as random instances of PERM – from the value of PERM(y_{ij}) over the integers, PERM(y_{ij}) mod p_j can be found simply by reducing mod p_j. In summary, the mapping from x to $\{y_{ij}\}$ is an rsr for PERM.

Let f be #P-complete. On input x of size n, the rsr for f proceeds as follows. Reduce x to one or more instances of PERM. Pad these instances if necessary so that their size depends only on n: For any $l \geq m$, an m-by-m matrix M can be "padded" out to an l-by-l matrix M' with the same permanent by letting $M'(a,b) = M(a,b)$, for $1 \leq a, b \leq m$, $M'(a,a) = 1$, for $m < a \leq l$, and $M'(a,b) = 0$, whenever $a \neq b$ and at least one of a and b is greater than m. Perform the above rsr of PERM. The random PERM-instances thus produced can be mapped back to f-instances, because f is #P-complete. These f-instances leak at most n, because the random PERM-instances leak at most n. ∎

The proof of Theorem 6.1 illustrates a general principle about the random-self-reducibility of two functions that are ptime-equivalent. It is not necessarily the case that if f and g are ptime-equivalent and g is rsr leaking at most n, then f is rsr leaking at most n. We might try to obtain an rsr for f by reducing an f-instance to one or more g-instances, random-self-reducing these g-instances, and then reducing the random g-instances back to f-instances. However, this overall reduction may leak more than the size of the original f-instance: The reduction from f to g may be such that it maps f-instances x_1 and x_2 of the same size to g-instances y_1 and y_2 of different sizes; if this so, it may be possible to tell whether we started with x_1 or x_2 after the instance y_i is randomized and mapped back to f-instances. However, if g is paddable, as the PERM function is in the above proof, this is not a problem – y_1 and y_2 can be padded so that their size depends only on the size of x_i. The general principle is that if f and g are ptime-equivalent and g is paddable and rsr leaking at most n, then f is rsr leaking at most n.

Corollary 6.2 [FF]: *All* PP*-complete sets are rsr.*

Proof: Let S be PP-complete. The language classes P^{PP} and $\mathrm{P}^{\#P}$ are equal. Thus S and PERM are ptime-equivalent. The rest of the proof is identical to that of Theorem 6.1. ∎

Theorem 6.3 [FF]: *All* PSPACE*-complete sets and all* EXP*-complete sets are rsr.*

Proof: Let $S \subseteq \{0,1\}^*$ be PSPACE-complete (resp. EXP-complete) and f_n be the restriction to $\{0,1\}^n$ of the characteristic function of S. Let $g = \{g_n\}_{n\geq 1}$ be the arithmetization of $f = \{f_n\}_{n\geq 1}$ over a suitable sequence of finite fields.

From Equations (1) and (2) in Section 4, it is clear that g is computable in polynomial space (resp. exponential time), because f is computable in polynomial space (resp. exponential time). (This requires that K_n not be too big, but this is not a problem, because we can always choose K_n to be of size polynomial in n.)

The rsr of S proceeds as follows. Let x be an element of $\{0,1\}^n$ for which we wish to determine membership in S. Interpret x as an element y of $Dom(g_n)$; this may be regarded as reducing f to g via the identity mapping. Map y to random g-instances y_1, ..., y_{n+1} using the standard rsr for low-degree polynomials given in Section 4. For each y_i, reduce the computation of $g(y_i)$ to a sequence z_{i1}, ..., z_{im} of S-oracle queries. This is a polynomial-time reduction, because S is PSPACE-complete (resp. EXP-complete) and g is computable in polynomial space (resp. exponential time). Overall, the mapping from x to the z_{ij}'s is an rsr for S.

Note that, because the reduction from f to g is the identity mapping, it is obvious that the overall reduction leaks at most n, and we need not worry about paddability of g. ∎

Feigenbaum and Fortnow [FF] have also shown that $\mathrm{MOD}_m\mathrm{P}$-complete sets are rsr, for every integer $m > 1$. ($\mathrm{MOD}_2\mathrm{P}$ is better known as $\oplus$P.)

Note that the overall reductions in Theorems 6.1 and 6.3 may be adaptive if f or S is Turing complete for #P, PSPACE or EXP but not truth-table complete. Similarly, the rsr for a PP-complete set given by the proof of Theorem 6.2 will be adaptive.

Question 6.4 *Are there natural examples of adaptive rsr's besides the ones for these complete sets and functions?*

The crucial fact in the proof of Theorem 6.3 is that if g is the arithmetization of f and f is computable in polynomial space (resp. exponential time), then so is g. Clearly, this is true of many other complexity classes, such as exponential space and double-exponential time. Feigenbaum and Fortnow give a precise sufficient condition on a complexity class under which the proof technique of Theorem 6.3 shows that complete sets or functions for that class are rsr – the condition is called #P-*robustness*; see [FF] for details. The proof of Theorem 6.1 does not use this technique; indeed it is not known whether #P is itself #P-robust.

Theorems 6.1, 6.2, and 6.3 tell us something remarkable about average-case complexity of complete sets (or functions) in #P, PP, PSPACE, and EXP. For each such complete set S, there is a polynomial-time sampleable distribution D on instances such that S must be hard on average with respect to D if it is hard (for randomized algorithms) in the worst case; of course these sets *are* believed to be hard in the worst case, because they are complete for their respective classes. Furthermore, the proofs of these average-case hardness results are quite simple. Average-case hardness results for NP are often quite tricky [Le].

Note that the rsr for the permanent over finite fields is a very direct reduction and produces uniformly distributed (although correlated) random instances. The reductions given by Theorems 6.1, 6.2, and 6.3 do not necessarily have these nice properties.

Question 6.5 *Find direct, simple rsr's for* #P*-complete functions of interest, such as the integer permanent function restricted to 0/1 matrices. Do the same for* PSPACE*-complete sets.*

We turn next to the consequences of the results of Sections 4 and 5 in the theory of multiprover, interactive proof systems.

Definition 6.6 [BGKW]: *A* **multiprover, interactive proof system for the set** $\boldsymbol{S}$ *is a collection* $(V, P_1, \ldots, P_k)$ *of shared-input, interactive Turing machines with the following properties (1) and (2). The probabilistic polynomial-time machine* V *is called the* **verifier***, and the powerful players* P_1*, …,* P_k *are called the* **provers***.*

(1) If $x \in S$*, then* $(V, P_1, \ldots, P_k)(x) = \text{ACCEPT}$ *with probability at least* $2/3$*.*

(2) If $x \notin S$, then for all provers P_1^*, ..., P_k^*, $(V, P_1^*, \ldots, P_k^*)(x) =$ REJECT with probability at least $2/3$.

The class of sets with multiprover, interactive proof systems is denoted MIP. A one-prover, interactive proof system [GMR] is one in which $k = 1$. The class of sets with one-prover, interactive proof systems is denoted IP. A function f has a multiprover (resp. one-prover), interactive proof system if its graph $\{(x, f(x))\}$ does. The class of functions with multiprover (resp. one-prover), interactive proof systems is denoted FMIP (resp. FIP). Note that the provers in a multiprover, interactive proof system are, by definition, allowed to toss coins. This ability is only needed, however, to obtain *zero-knowledge* proof systems [GMR, BGKW]. For the rest of this section, we may assume without loss of generality that the provers are deterministic; they may still be adaptive, i.e., they may still base their answers in round l on all of the verifier's queries in rounds 1 through $l - 1$.

Theorem 6.7 [FRS]: *A set S (resp. function f) is in* MIP *(resp.* FMIP*) if and only if it has a one-prover, interactive proof system in which both the legitimate prover P and any cheating prover P^* are functional.*

Sketch of proof: Suppose that $(V, P_1, \ldots, P_k)$ is an r-round, k-prover, interactive proof system for S. We construct an rk-round, one-prover system (V_0, P_0) with the desired properties.

Let $(a_{i1}, \ldots, a_{ir})$, $1 \le i \le k$, be the sequence of queries that V sends to P_i during the execution of $(V, P_1, \ldots, P_k)$ on input x and random string *ran*. With the same input and random string, in round $lk + i$, $0 \le l \le r - 1$, V_0 sends to P_0 the query $(a_{i1}, \ldots, a_{i,l+1})$. P_0 answers exactly as P_i would have in the multiprover system. At the end of all of the interaction, V_0 has precisely the information that V has at the end of the interaction in the multiprover system, and V_0 accepts if and only if V would have.

If $x \in S$, then the legitimate P_0 obviously convinces V_0 to accept with exactly the same probability that P_1, ..., P_k convince V to accept. Suppose that a cheating P_0^* could convince V_0 to accept an x that is not in S. Recall that P_0^* is functional. So his successful dishonest response in round $lk + i$ must be based on the query $(a_{i1}, \ldots, a_{i,l+1})$ and *not* on queries received in other rounds. Thus, we could construct cheating provers P_1^*, ..., P_k^* who would succeed in making V accept x, because $(a_{i1}, \ldots, a_{i,l+1})$ is exactly the information that the i^{th} prover has in round $l+1$. This is a contradiction, because $(V, P_1, \ldots, P_k)$ is a sound multiprover system for S.

Suppose S is accepted by the n^k-round, one-prover system (V_0, P_0), and that both P_0 and any cheating prover P_0^* are functional. We define a multiprover system $(V, P_1, \ldots, P_{n^{k+1}})$ that accepts S. The verifier V first randomly chooses an ordering of the n^{k+1} provers. V then simulates V_0 and whenever V_0 asks a question to P_0, V asks the same question to each of the next n provers in the chosen ordering. If the provers are unanimous in their answer, V uses that answer in its simulation of V_0; if not, V rejects immediately. If the provers achieve unanimity on all queries, then V accepts if and only if V_0 does.

A complete proof that $(V, P_1, \ldots, P_{n^{k+1}})$ is a correct multiprover system for S can be found in [Fo]. ∎

Note that the fact that the one-prover system is only secure against functional cheating provers gives it the power to recognize a set that may not be in IP – in IP, adaptive cheating provers must also be defeated.

Checker characterization theorem [BK]: A function is checkable if and only if it has a one-prover, interactive proof system in which the legitimate prover P computes f and any cheating prover P^* is functional.

Thus, if a set (resp. function) is checkable, it is in MIP (resp. FMIP).

The connection between random-self-reducibility and FMIP, via checkability, gives a simple, high-level proof that large, interesting classes have multiprover, interactive proof systems. If we use Theorem 5.4 to show that f has a self-testing/correcting pair, then we have also shown that $f \in$ FMIP. If f is hard for complexity class C, then we have shown that $C \subseteq$ FMIP (but *not* that every function in C is checkable!). For example, it is immediate that all of #P is in FMIP: From the proof of Theorem 6.1, it is clear that the #P-complete PERM function has a size-preserving rsr; PERM also has a downward self-reduction, which consists of expansion by co-factors. This observation played an important role in the complete characterization of the power of interactive proof systems: IP = PSPACE (cf. [LFKN, S2]), and MIP = NEXP (cf. [BFL]). The random-self-reducibility of low-degree, multivariate polynomials also played a crucial role in the investigation of the power of two-prover, *one-round* interactive proof systems. A sequence of results culminating in those of Feige and Lovász show that these MIP$(2,1)$ proof systems recognize all of NEXP; [FL, Sections 1 and 2] gives the history of this problem and a proof of the final result.

One way to interpret the fact that PSPACE-complete sets (or PP-complete sets or #P-complete functions) are checkable is that, in a multiprover, interactive proof system for such a set, the provers need only sufficient power to decide membership in the set itself. (This is also true of EXP-complete sets, but not by the above argument – these sets are not downward self-reducible, unless EXP = BPPSPACE, which is in fact equal to PSPACE. See [BFL] for a proof that EXP-complete sets have self-testing/correcting pairs.) It is not known whether coNP-complete sets have multiprover, interactive proof systems in which the provers need only sufficient power to determine membership in the set itself; we will return to this issue in the next section.

7 Negative Results

We now review the known limitations on random-self-reductions, locally random reductions, and instance-hiding schemes. There are many interesting open questions in this area.

The first general negative result was proven by Abadi, Feigenbaum, and Kilian in the original paper on instance-hiding.

Theorem 7.1 [AFK]: *If S has a 1-ihs, then $S \in \mathrm{NP}/poly \cap \mathrm{coNP}/poly$.*

Sketch of proof: First suppose that (A, B) is an r-round 1-ihs that has error probability 0. Thus, for any input x, player A always gets the right answer $\chi_S(x)$, no matter which random string ran he uses.

For each input size n, choose an input x_n in I_n and a random string $ran_n \in \{0,1\}^{w(n)}$. Let $t_n = (a_1, b_1, \ldots, a_{r(n)}, b_{r(n)})$ be a transcript that players A and B could produce on input string x_n and random string ran_n. For every other x of size n, there must be at least one random string ran such that transcript t_n is produced. Otherwise, the ihs would leak more than n.

The nonuniform, nondeterministic polynomial-time algorithm for χ_S is now obvious. The advice for size n is t_n. On input $x \in I_n$, the algorithm guesses $ran \in \{0,1\}^{w(n)}$ such that (A, B) produces t_n, simulates the ihs, and returns the output of A.

If the ihs (A, B) is allowed to make errors, the advice must be more elaborate. For example, there may not be a single transcript t_n with the property that, for every $x \in I_n$, there is at least one ran such that A outputs the right answer $\chi_S(x)$ on input x, random string ran and transcript t_n. We must show that there is a polynomial-sized *set* of transcripts, say T_n, such that, for any $x \in I_n$, the majority of transcripts in T_n produce the right answer on input x for the overwhelming majority of random strings ran. Furthermore, since NP/*poly* $\cap$ coNP/*poly* is not a probabilistic class, we need a way of verifying that a particular ran is in this overwhelming majority. Both of these constructions are given in full detail in [AFK]. ∎

Corollary 7.2 *If any* NP*-hard function has a 1-ihs, then the polynomial-time hierarchy collapses at the third level.*

This follows directly from Theorem 7.1 and from the fact that the hierarchy collapses at the third level if NP is in coNP/*poly* (cf. [Yap]). Note that the hierarchy collapses at the third level if any NP-hard function has a 1-rsr or a 1-lrr, because these are just special cases of 1-ihs's.

The question of whether NP-hard functions have k-ihs's, for some $k > 1$, was posed by Rivest. We have already seen that the answer to this question is yes: Theorem 4.1 states that *every* boolean function on n variables has a nonadaptive $(n+1)$-lrr, which is a special type of $(n+1)$-ihs. An ihs for a non-boolean function f can be obtained by concatenating the ihs's for each output bit.

Beaver and Feigenbaum's general construction shows that there is an ihs for every function f. However, this construction does not answer the question of how much computational power must be used to answer the random questions. For example, in an lrr from f to g, how does the computational complexity of g compare with that of f? As discussed in Section 6, sets (or functions) complete for the complexity classes #P, PP, PSPACE, and EXP have rsr's. Thus, in those cases, the complexity of g is the same as that of f. What about complete sets of lower complexity, e.g., NP-complete sets? Feigenbaum and Fortnow have made some progress on this question.

Theorem 7.3 [FF]: *If any* NP*-complete set is nonadaptively rsr, then the polynomial-time hierarchy collapses at the third level.*

Sketch of proof: We use the notation AM(r) to denote the class of sets accepted by r-move Arthur-Merlin games [BM]. The prover Merlin is denoted by P and the verifier Arthur by V; a cheating prover is denoted by P^*. The class AM(r)/*poly* consists of all sets accepted by r-move Arthur-Merlin games in which the verifier is given nonuniform, polynomial-size advice as well as probabilistic polynomial time.

Let S be NP-complete. We will show that if S is nonadaptively rsr, then $\overline{S}$ (and hence all of coNP) is in AM/*poly*. To complete the proof, recall that AM/*poly* = NP/*poly* (cf. [GS]) and that the hierarchy collapses at the third level if coNP is in NP/*poly* (cf. [Yap]).

Let σ, ϕ be a nonadaptive k-rsr for S, where $k = k(n)$ is a polynomially bounded function. As usual, we may assume without loss of generality that ϕ gives an incorrect value for the characteristic function of S with probability at most 2^{-n}.

Consider instances x of size n. The verifier's advice is the k-tuple $(p_1, \ldots, p_k)$, where p_i is the probability that a target instance $\sigma(i, x, ran)$ is in S. The probability is computed over all coin-toss sequences ran.

We denote by $Tr(x, ran)$ the vector $(y_1,\ b_1,\ \ldots,\ y_k,\ b_k)$, where $y_i = \sigma(i, x, ran)$ and $b_i = \chi_S(y_i)$. Fix a specific NP machine M that accepts S. Then $ATr(x, ran)$ denotes $(y_1, b_1, w_1, \ldots, y_k, b_k, w_k)$, where y_i and b_i are as before, $w_i =$ NIL if $b_i = 0$, and w_i is a witness, with respect to M, that $y_i \in S$ if $b_i = 1$.

The following is an AM/*poly* protocol for $\overline{S}$. Let m be a suitably large polynomial in k.

Arthur-Merlin game for $\overline{\mathbf{S}}$:

The quantifiers "for all $1 \leq i \leq k$" and "for all $1 \leq j \leq m$" are implicit whenever the subscripts i and j are used. For each $j \neq j'$, ran_j is independent of $ran_{j'}$.

V: Choose ran_j.

$V \rightarrow P$: $\{ran_j\}$.

$P \rightarrow V$: A claimed value $(y_{1,j}, b_{1,j}, w_{1,j}, \ldots, y_{k,j}, b_{k,j}, w_{k,j})$ for $ATr(x, ran_j)$.

V: Accept iff

(1) $\phi(x, ran_j, b_{1,j}, \ldots, b_{k,j}) = 0$,

(2) More than $p_i m - 2\sqrt{km}$ of the $y_{i,j}$'s are in S according to P, and

(3) If $w_{i,j} \neq$ NIL, then it is a correct witness that $y_{i,j} \in S$.

It is quite straightforward to show that, if $x \in \overline{S}$ and P is honest, then V accepts with probability at least $2/3$. So suppose that $x \in S$ and that a cheating prover P^* tries to convince V to accept x anyway. In order to succeed, P^* must send V messages that satisfy condition (1). Thus, either there is a j such that $\phi(x, ran_j, b_{1,j}, \ldots, b_{k,j})$ is wrong when all $b_{i,j}$ are correct, or P^* must lie about at least one $b_{i,j}$ for each j. The probability that there is a j such that $\phi(x, ran_j, b_{1,j}, \ldots, b_{k,j})$ is wrong when all $b_{i,j}$ are correct is exponentially small; so we must argue that, if P^* lies about at least one $b_{i,j}$ for each j, the probability that he meets acceptance condition (2), which he must do in order to get V to accept, is also sufficiently small. If he lies about at least one $b_{i,j}$ for each j, then he tells a total of at least m lies, and there must be a particular i, say i_0, for which he lies about at least m/k of the $b_{i_0,j}$'s. Because he is required to provide a witness if he claims that $y_{i_0,j} \in S$, he can only lie by claiming that something is not in S when it really is. Thus the only way he can meet condition (2) for i_0 is if the real number of $y_{i_0,j}$'s that are in S is more than $p_i m + m/k - 2\sqrt{km}$. Chernoff bounds suffice to show that the probability that Z_i exceeds its expected value by more than $m/k - 2\sqrt{km}$ is sufficiently small.

Full details of this proof can be found in [FF]. ∎

Feigenbaum and Fortnow also consider adaptive rsr's for complete sets in the polynomial-time hierarchy. Their definition of adaptive rsr's is slightly more restrictive than the one we use here. In their definition, the querier asks each of the powerful players exactly one random question. However, the negative result that they obtain is valid for our less restrictive definition as well: If f has an r-round, adaptive k-rsr under our definition, then f also has an r-round, adaptive rk-rsr under the definition in [FF]. In the following theorem, the number of powerful players k is an arbitrary polynomial, and thus the two definitions are equivalent for purposes of the theorem.

Theorem 7.4 [FF]: *If any* NP*-complete set has an* $O(\log n)$*-round, adaptive rsr, then the polynomial-time hierarchy collapses at the third level.*

The proof of Theorem 7.4 has the same structure as that of Theorem 7.3. The heart of the argument is to show that if an NP-complete set has an $O(\log n)$-round, adaptive rsr, then coNP is in AM/*poly*. The argument is trickier than in the nonadaptive case, because it is no longer true that a cheating prover can only lie by claiming that something is not in the set when it really is. By giving wrong answers to queries in earlier rounds of the reduction, the cheating prover may be able to force queries in later rounds to have the answers he wants. As long as $r = O(\log n)$, we can still design the proof system so that the verifier will catch a prover who tries to do this; see [FF] for a more detailed explanation. Unfortunately, this technique does not say anything about adaptive rsr's in which the number of rounds is not $O(\log n)$.

Question 7.5 *Do* NP*-complete sets have adaptive rsr's?*

Part of the motivation for studying the question of whether NP-complete sets have rsr's is Theorem 5.4. These sets do have downward self-reductions; so if they had size-preserving rsr's, they would have checkers, as do the complete sets for PP, PSPACE, and EXP and the complete functions for #P. Of course, they might have checkers even if they do not have rsr's.

Question 7.6 *Are* NP*-complete sets checkable? More generally, if S is NP-complete, how powerful must the provers be in a multiprover, interactive proof system for $\overline{S}$? How powerful must the single prover be in a one-prover, interactive proof system for $\overline{S}$?*

Question 7.7 *Do* NP*-complete sets have self-testing/correcting pairs?*

Analogues of theorems 7.3 and 7.4 hold for complete sets higher up in the polynomial hierarchy, i.e., if S is complete for Σ_i^p, and S has an $O(\log n)$-round, adaptive rsr, then the hierarchy collapses at the $(i+2)^{nd}$ level. Similarly, questions 7.5, 7.6, and 7.7 are valid for complete sets higher up in the hierarchy.

These questions are also valid for sets in the hierarchy that are not complete. The following result is due to Beigel, Bellare, Feigenbaum, and Goldwasser. The proof given here is more concise than the one in [BBFG] but uses the same basic idea. The complexity class EE consists of all sets in DTIME($2^{2^{O(n)}}$). NEE and BPEE denote the corresponding nondeterministic and bounded-probabilistic time classes.

Theorem 7.8 [BBFG]: *If* NEE $\not\subseteq$ BPEE*, then there is an uncheckable set in* NP*.*

Proof: Assume that NEE $\not\subseteq$ BPEE. A standard padding argument [Bo] shows that there is a set $S \in$ NP $\setminus$ BPP such that $S \subset \{0^{2^l} : l \geq 0\}$. We show that if S were checkable, it would be in BPP.

Suppose that Ch were a checker for S. Assume without loss of generality that Ch's error probability is exponentially small. There is a fixed polynomial p such that, on inputs of size n, Ch only queries the oracle about strings of size at most $p(n)$. Because S is so sparse and has such a simple structure, a bit vector L of length $\log(p(n))$ can completely describe all members of S that have size at most $p(n)$.

The BPP machine N for S works as follows on input x of size n. Let $L_1, \ldots, L_{p(n)}$ be all of the (polynomially many!) possibilities for L. For each L_i, N simulates Ch on input x, answering all oracle queries according to L_i. As soon as N reaches a value of i for which Ch outputs CORRECT, N either accepts or rejects x, depending on whether L_i says x is in S or not. If none of the simulations causes Ch to output CORRECT (a low-probability event by the definition of checkability), then N rejects x. ∎

Corollary 7.9 *If* NEE $\not\subseteq$ BPEE*, then there is a set in* NP *that does not have a self-testing/correcting pair.*

Theorem 7.1 shows that there exist functions that do not have 1-ihs's, even if multiple rounds of queries are allowed. Theorem 4.3 shows that every function has a $(1 + (n/\log n))$-ihs, even if only one round of queries is allowed. There is obviously a big gap between these results – perhaps every function has a 2-ihs. In fact, no one has shown that this is not the case. Fortnow and Szegedy, building on previous work of Yao, have obtained the following partial result on the limitations of 2-ihs's.

Theorem 7.10 [FS]: *If S is nonadaptively 2-lrr to $(g1, g2)$, where $g1$ and $g2$ are boolean functions and the 2-lrr has error probability 0, then $S \in \mathrm{NP}/poly \cap \mathrm{coNP}/poly$.*

Sketch of proof: Let σ, ϕ be an errorless 2-lrr from χ_S to boolean functions $g1$, $g2$. If $y_i = \sigma(i, x, ran)$ and $z_i = gi(y_i)$, $i = 1, 2$, we say that x, ran, and z_i *set* $\chi_S(x)$ if the value of z_{3-i} is not needed by ϕ in computing $\chi_S(x)$. That is, $\phi(x, ran, z_i, 0) = \phi(x, ran, z_i, 1) = \chi_S(x)$. We say that z_i *forces* z_{3-i} *through* x *and* ran if x, ran, and z_i do not set $\chi_S(x)$. Note that the assumption that z_1 and z_2 are single bits in necessary for the notion of forcing to be well-defined.

Let $T = \{x_1, \ldots, x_t\}$ be a set of inputs in I_n. The sequence y_0, y_1, …, y_m is a *forcing path with respect to* T if: There exist $x_0 \in T$ and ran_0 such that $y_0 = \sigma(1, x_0, ran_0)$, $y_1 = \sigma(2, x_0, ran_0)$, and $g1(y_0)$ forces $g2(y_1)$ through x_0 and ran_0; there exist $x_1 \in T$ and ran_1 such that $y_1 = \sigma(2, x_1, ran_1)$, $y_2 = \sigma(1, x_1, ran_1)$, and $g2(y_1)$ forces $g1(y_2)$ through x_1 and ran_1; there exist $x_2 \in T$ and ran_2 such that $y_2 = \sigma(1, x_2, ran_2)$, $y_3 = \sigma(2, x_2, ran_2)$, and $g1(y_2)$ forces $g2(y_3)$ through x_2 and ran_2; and so forth. That is, the appropriate g-values are all forced by $g1(y_0)$ and by values of χ_S on elements of T. The *description* of the forcing path consists of y_0, y_1, …, y_m and the sequence of x_j, ran_j pairs.

On inputs in I_n, the nondeterministic, polynomial-time algorithm for χ_S will use as advice a starting point $g1(y_0)$, a polynomial-sized subset T of I_n, and the values of χ_S at elements of T. On input x, the algorithm will guess a random string ran and a description of a forcing path with respect to T that starts at y_0 such that

(1) $z_1 = g1(\sigma(1, x, ran))$ and $z_2 = g2(\sigma(2, x, ran))$ are both on this forcing path, or

(2) z_i is on the path and $\chi_S(x)$ is set by x, ran, and z_i.

It is clear how $\chi_S(x)$ can be obtained by the algorithm but why must such a polynomial-sized set T exist? Fortnow and Szegedy give a recursive construction of T. The crux of their construction is the following expansion property. Suppose that $T_j = \{x_1, \ldots x_j\}$ and that $U_j \subseteq Range(\sigma)$ is such that every point in U_j is on a forcing path with respect to T_j of length at most j. Let x_{j+1} be an input in I_n for which neither (1) nor (2) holds, and take T_{j+1} to be $T_j \cup \{x_{j+1}\}$. Suppose ran is a random string such that exactly one of $y_1 = \sigma(1, x_{j+1}, ran)$ and $y_2 = \sigma(2, x_{j+1}, ran)$ is in U_j. If $y_i \in U_j$, then y_{3-i} is forced by x_{i+1}, ran, and $gi(y_i)$. Let U_{j+1} consist of all y_1's and y_2's obtained in this manner. Then $|U_{j+1}| = 2|U_j|$, and every point in U_{j+1} lies on a forcing path with respect to T_{j+1} of length at most $j+1$. If we repeat this doubling process $t = poly(n)$ times, we can take $T = T_t$ and have $U_t = Range(\sigma)$.

All of the details can be found in [FS]. ∎

Corollary 7.11 *If any* NP*-hard set is nonadaptively 2-lrr to* $(g1, g2)$*, where* $g1$ *and* $g2$ *are boolean functions and the 2-lrr has error probability 0, then the polynomial-time hierarchy collapses at the third level.*

8 Related Work

In conclusion, we discuss briefly several notions in interactive complexity theory that are related to those we've covered in more detail and give pointers to the relevant papers.

The properties of an instance-hiding scheme and those of an interactive proof system can be combined in one set of interactive Turing machines. Such an *instance-hiding proof system* may also have the zero-knowledge property. Informally, a *zero-knowledge, instance-hiding proof system* is a set of private-input, interactive Turing machines in which the verifier is convinced of the truth of a statement, but the verifier does not learn the proof, and the provers do not learn what they are proving. Beaver, Feigenbaum, and Shoup formalize this notion and show that every set in NEXP has a perfect zero-knowledge, multiprover, instance-hiding proof system [BFS]. Under the assumption that one-way permutations exist, Feigenbaum and Ostrovsky show that any set that has both a one-prover, interactive proof system and a 1-ihs in fact has a zero-knowledge, one-prover, instance-hiding proof system [FO]. Blum, Luby, and Rubinfeld consider *private, adaptive checkers*, which are instance-hiding proof systems for f in which the legitimate prover is functional and is required to compute f but the cheating prover may be adaptive [BLR2]. They show that any function that is rsr and downward self-reducible has a private, adaptive checker.

Yao defined the notion of a *coherent function* [Yao]. An *examiner* M for a function

f is a probabilistic polynomial-time oracle machine such that M^f computes f and, on input x, never queries the oracle about x. A function f is coherent if it has an examiner. Beigel, Bellare, Feigenbaum, and Goldwasser extend this notion to nonuniform examiners: A function is *weakly coherent* if it has an examiner that is given polynomial advice as well as probabilistic polynomial-time [BBFG]. If f is checkable, it is coherent [Yao], and it if is rsr, it is weakly coherent [BBFG]. These connections are used in [BBFG] to construct a function f that is uncheckable and non-rsr whose space complexity is slightly more than polynomial, i.e., $f \in \mathrm{FSPACE}(n^{\log^* n})$. This is the best bound on such a function that has been proven without any complexity-theoretic assumptions.

Feigenbaum, Fortnow, Lund, and Spielman [FFLS] show that increasing the number of random queries makes random-self-reductions strictly more powerful; specifically, they show that there are functions that are nonadaptively k-rsr but not adaptively $(k-1)$-rsr. They also show that adaptive rsr's are strictly more powerful than nonadaptive rsr's, i.e., that there is a function that is not nonadaptively rsr but is adaptively rsr. Neither of these results relies on a complexity-theoretic assumption. Under the assumption that NEEE $\not\subseteq$ BPEEE, they show that there is a set in NP that is not nonadaptively rsr but is adaptively rsr.

Finally, the notions of random-self-reducibility and self/testing-correction play a role in the recent, exciting results on *probabilistically checkable proofs* [AS, ALMSS, FGLSS, LY]. By showing that all sets in NP have proof systems of a very restricted, communication-efficient form, one can conclude that certain classical optimization problems, such as chromatic number and clique number, cannot be solved *approximately* in polynomial time unless they can be solved *exactly* in polynomial time, i.e., unless P = NP.

9 Acknowledgements

I thank Lance Fortnow and Carsten Lund for many useful and enjoyable discussions of this material and for their comments on an earlier draft of the paper. Helpful comments were also provided by Sigal Ar, Toni Pitassi, and an anonymous referee.

References

[AFK] M. Abadi, J. Feigenbaum, and J. Kilian. *On Hiding Information from an Oracle*, J. Comput. System Sci. **39** (1989), 21–50.

[AL] D. Angluin and D. Lichtenstein. *Provable Security of Cryptosystems: a Survey*, Yale University Technical Report YALEU/DCS/TR-288, New Haven CT, 1983.

[ALMSS] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. *Proof Verification and Hardness of Approximation Problems*, IEEE Symposium on Foundations of Computer Science **33** (Pittsburg, PA; October, 1992).

[AS] S. Arora and S. Safra. *Probabilistic Checking of Proofs*, IEEE Symposium on Foundations of Computer Science **33** (Pittsburg, PA; October, 1992).

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

[BFL] L. Babai, L. Fortnow, and C. Lund. *Nondeterministic Exponential Time has Two-Prover Interactive Protocols*, Comput. Compl. **1** (1991), 3–40.

[BM] L. Babai and S. Moran. *Arthur-Merlin Games: A Randomized Proof System and a Hierarchy of Complexity Classes*, J. Comput. System Sci. **36** (1988), 254–276.

[BeaF] D. Beaver and J. Feigenbaum. *Hiding Instances in Multioracle Queries*, Symposium on Theoretical Aspects of Computer Science **7** (Rouen, France; February, 1990), Springer Verlag LNCS 415, 37–48.

[BFKR] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. *Security with Low Communication Overhead*, Advances in Cryptology – Crypto '90 (Santa Barbara, CA; August, 1990), Springer Verlag LNCS 537, 62–76.

[BFS] D. Beaver, J. Feigenbaum, and V. Shoup. *Hiding Instances in Zero-Knowledge Proof Systems*, Advances in Cryptology – Crypto '90 (Santa Barbara, CA; August, 1990), Springer Verlag LNCS 537, 326–338.

[BBFG] R. Beigel, M. Bellare, J. Feigenbaum, and S. Goldwasser. *Languages that are Easier than Their Proofs*, IEEE Symposium on Foundations of Computer Science **32** (San Juan, PR; October, 1991), 19–28.

[BGKW] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. *Multiprover Interactive Proof Systems: How to Remove Intractability Assumptions*, ACM Symposium on Theory of Computing **20** (Chicago, IL; May, 1988), 113–131.

[Bl] M. Blum. *Designing Programs to Check Their Work*, Intl. Comput. Sci. Inst. Technical Report 88-009, Berkeley CA, 1988.

[BK] M. Blum and S. Kannan. *Designing Programs that Check Their Work*, ACM Symposium on Theory of Computing **21** (Seattle, WA; May, 1989), 86–97.

[BLR1] M. Blum, M. Luby, and R. Rubinfeld. *Self-Testing/Correcting with Applications to Numerical Problems*, ACM Symposium on Theory of Computing **21** (Baltimore, MD; May, 1990), 73–83.

[BLR2] M. Blum, M. Luby, and R. Rubinfeld. *Program Result Checking against Adaptive Programs and in Cryptographic Settings*, in Distributed Computing and Cryptography, DIMACS Series in Discrete Mathematics and Theoretical Computer Science **2** (1991), AMS, 107–118.

[BlMi] M. Blum and S. Micali. *How to Generate Cryptographically Strong Sequences of Pseudorandom Bits*, SIAM J. Comput. **13** (1984), 850–864.

[Bo] R. Book. *Tally Languages and Complexity Classes*, Inf. and Control **26** (1974), 186–193.

[FGLSS] U. Feige, S. Goldwasser, L. Lovász, M. Safra, and M. Szegedy. *Approximating Clique is Almost NP-Complete*, IEEE Symposium on Foundations of Computer Science **32** (San Juan, PR; October, 1991), 2–12.

[FL] U. Feige and L. Lovász. *Two-Prover One-Round Proof Systems: Their Power and Their Problems*, ACM Symposium on Theory of Computing **24** (Victoria, BC; May 1992), 733–744.

[FF] J. Feigenbaum and L. Fortnow. *On the Random-Self-Reducibility of Complete Sets*, SIAM J. Comput., to appear. Preliminary version in IEEE Structure in Complexity Theory Conference **6** (Chicago, IL; July, 1991), 124–132.

[FFLS] J. Feigenbaum, L. Fortnow, C. Lund, and D. Spielman. *The Power of Adaptiveness and Additional Queries in Random-Self-Reductions*, IEEE Structure in Complexity Theory Conference **7** (Boston, MA; June, 1992), 338–346.

[FO] J. Feigenbaum and R. Ostrovsky. *A Note on One-Prover, Zero-Knowledge, Instance-Hiding Proof Systems*, Advances in Cryptology – Asiacrypt '91 (Fujiyoshida, Japan; November, 1991), Springer Verlag LNCS, to appear.

[Fo] L. Fortnow. *Complexity-Theoretic Aspects of Interactive Proof Systems*, PhD Thesis, Mass. Inst. of Tech. Technical Report MIT/LCS/TR-447, Cambridge MA, 1989.

[FK] L. Fortnow and S. Kurtz. Private communication, December, 1991.

[FRS] L. Fortnow, J. Rompel, and M. Sipser. *On the Power of Multiprover Interactive Protocols*, IEEE Structure in Complexity Theory Conference **3** (Washington DC; June, 1988), 156–161.

[FS] L. Fortnow and M. Szegedy. *On the Power of Two-Locally Random Reductions*, Advances in Cryptology – Asiacrypt '91 (Fujiyoshida, Japan; November, 1991), Springer Verlag LNCS, to appear.

[GM] S. Goldwasser and S. Micali. *Probabilistic Encryption*, J. Comput. System Sci. **28** (1984), 270–299.

[GMR] S. Goldwasser, S. Micali, and C. Rackoff. *The Knowledge Complexity of Interactive Proof Systems*, SIAM J. Comput. **18** (1989), 186–208.

[GS] S. Goldwasser and M. Sipser. *Public Coins vs. Private Coins in Interactive Proof Systems*, Advances in Computing Research – Vol. 5: Randomness and Computation, JAI Press, Greenwich, 1989, 73–90.

[Ka] S. Kannan. *Program Checkers for Algebraic Problems*, PhD Thesis, Intl. Comput. Sci. Inst. Technical Report 89-064, Berkeley CA, 1989.

[Le] L. Levin. *Average Case Complete Problems*, SIAM J. Comput. **15** (1986), 285–286.

[Li] R. Lipton. *New Directions in Testing*, in Distributed Computing and Cryptography, DIMACS Series in Discrete Mathematics and Theoretical Computer Science **2** (1991), AMS, 191–202.

[LFKN] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. *Algebraic Methods for Interactive Proof Systems*, IEEE Symposium on Foundations of Computer Science **31** (St. Louis, MO; October, 1990), 2–10.

[LY] C. Lund and M. Yannakakis. *On the Hardness of Approximating Minimization Problems*, manuscript, July 1992.

[Ru] R. Rubinfeld. *A Mathematical Theory of Self-Checking, Self-Testing, and Self-Correcting Programs*, PhD Thesis, Intl. Comput. Sci. Inst. Technical Report 90-054, Berkeley CA, 1990.

[S1] A. Shamir. *How to Share a Secret*, Comm. Assoc. Comput. Machinery **22** (1979), 612–613.

[S2] A. Shamir. $IP = PSPACE$, IEEE Symposium on Foundations of Computer Science **31** (St. Louis, MO; October, 1990), 11–15.

[Yao] A. C. Yao. *Coherent Functions and Program Checkers*, ACM Symposium on Theory of Computing **22** (Baltimore, MD; May, 1990), 84–94.

[Yap] C. Yap. Some Consequences of Nonuniform Conditions on Uniform Classes, *Theor. Comput. Sci.* **26** (1983), 287–300.