Stronger separations for random-self-reducibility, rounds, and advice

László Babai
Computer Science Department
University of Chicago
Chicago, IL 60637, USA
laci@cs.uchicago.edu

Sophie Laplante
LRI, Université Paris-Sud
91405 Orsay, France
laplante@lri.fr

Abstract

A function f is self-reducible if it can be computed given an oracle for f. In a random-self-reduction *the queries must be made in such a way that the distribution of the ith query is independent of the input that gave rise to it. Random-self-reductions have many applications, including countless cryptographic protocols, probabilistically checkable proofs, average-case complexity, and program checking. A simpler model of randomized self-reducibility is coherence, in which the only condition on the queries is that the input itself may not be among the queries. We show that there is a function which is random-self-reducible with 2 rounds of queries, but which is not even coherent, even if polynomial advice is allowed, when the queries must be made in a single round.*

1 Introduction

Informally, we say that a function f self-reduces if it can be computed efficiently by making queries to f. For a function to be *random-self-reducible*, the queries must be made at random in such a way that the distribution of an individual query does not give away any information on the input, except perhaps its length.

The notion of random-self-reducibility is central to many areas of theoretical computer science. Initially, it emerged as a formalization of a notion used in countless cryptographic protocols and interactive proofs [2, 6, 13, 14]. It is also used in the context of probabilistically checkable proofs, program checkers, self-testers and correctors, and has applications in average-case complexity. (See Feigenbaum's survey paper for a description of these and other applications [5].)

Random-self-reducibility has also turned out to be an important structural property in complexity. It is known that all complete sets for **PSPACE**, **EXP**, as well as some other classes above the polynomial hierarchy are random-self-reducible. On the other hand, sets complete for the kth level of the polynomial hierarchy cannot be nonadaptively random-self-reducible unless the hierarchy collapses at the $k+2$nd level. The question of whether these sets are adaptively random-self-reducible remains open. (See the paper by Feigenbaum and Fortnow [6] for these and related results.)

More recently, the work of Buhrman, Fortnow, van Melkebeek and Torenvliet [4] shows how to use autoreducibility, a deterministic version of self-reducibility, to separate complexity classes.

In an attempt to capture the weakest possible notion of randomized self-reductions, Yao [15] introduced coherence. A function is coherent if it can be computed by a probabilistic polynomial-time Turing machine which can query the function on any point but the input. Every random-self-reducible set is also coherent with polynomial advice [3].

It is natural to consider self-reductions in which the queries can be made adaptively, that is, in several rounds, or nonadaptively (in parallel.)

A few results are known about the relationship between the various flavors of self-reductions. Feigenbaum, Fortnow, Lund and Spielman [8] have shown that there are functions which are adaptively random-self-reducible but not nonadaptively random-self-reducible. (Specifically, they show that if **NEEE** $\not\subseteq$ **BPEEE** then such a function exists in **NP** $\setminus$ **P**.) Hemaspaandra, Naik, Ogihara and Selman [9] separate adaptive random-self-reductions from nonadaptive deterministic self-reductions. (They show that if **NP** $\not\subseteq$ **BPE** then there is a set in **NP** $\setminus$ **BPP** which is adaptively random-self-reducible, but not nonadaptively random-self-reducible, and not self-reducible.) Feigenbaum, Fortnow, Laplante and Naik [7] give the first lower bound for a nonuniform model: they show that there are functions which are adaptively coherent but not nonadaptively coherent even with polynomial advice. (This result is unconditional but there is no upper bound on the complexity of the function, as in the two previous results.) Laplante [11] extends this result to show that for every constant k, there is a function that is $k+1$-round coherent but not k-round coherent even with polynomial advice.

In this paper we are able to improve the results cited above [8, 7], and separate multi-round random-self-reducible functions from single-round coherent functions which take polynomial advice.

The technique used to show the main result is based on Kolmogorov complexity. The proof does not involve any complex probability arguments, there are no issues of independence of distributions, and nonuniformity is handled very easily. This contrasts with the usual techniques, such as diagonalization and probabilistic arguments. This approach could be extended to give similar lower bounds for program checking and other applications of random-self-reductions.

2 Preliminaries

We start by giving some definitions that we will use in the paper.

2.1 Randomized self-reductions

We use two randomized notions of self-reducibility. The simplest of the two is coherence, a bare-bones notion of randomized self-reducibility.

Definition 1 *A function is* **coherent** *if it can be computed by a probabilistic polynomial-time Turing machine which can query the function on any point but the input itself. If the Turing machine which computes the function in this way takes polynomial advice (following Karp and Lipton [10]) then the function is* **coherent with advice**.

The second notion we use is random-self-reducibility, in which a strong restriction is made on the way the queries that are made, so that the input is not revealed by the queries' distribution.

Definition 2 *A function is* **random-self-reducible** *if it can be computed by a probabilistic polynomial-time Turing machine which may query the function on any point but with an additional constraint on the distribution of the queries. The distribution of each query must be independent of the input that it originated from.*

Note that the distribution of individual queries is independent of the input, but the joint distribution of the queries may be (highly) correlated with the input.

If the queries are made in a single round, then the function is called **nonadaptively random-self-reducible**. If they are made in several rounds, then the function is **adaptively random-self-reducible**.

We would like to compare random-self-reducibility with a weaker model in which the queries are not required to hide the input. However, it is not necessarily the case that every random-self-reducible function is coherent. This is because in a random-self-reduction, it is possible that the input itself gets queried. In fact, if this were never the case, then the distribution of queries would be skewed away from the input, which would yield information about the input. On the other hand, a random-self-reduction cannot query its own input often – again the distribution would be skewed. To transform an arbitrary random-self-reduction into a coherent reduction, one may have to encode the value of the function on the (few) inputs on which the input is one of the queries. This is the main idea behind the proposition below.

Proposition 1 *[3] Every nonadaptively random-self-reducible function is nonadaptively coherent with polynomial advice.*

For this reason, we compare random-self-reducibility with nonuniform coherence.

2.2 Multilinear extensions

The "low-degree polynomial trick," or "arithmetization" is a standard way to construct functions that are random-self-reducible. This hinges on the following fact: any family of multivariate polynomials of degree polynomially bounded in the number of variables is random-self-reducible.

Fix a finite field K of order at least $n+1$ ($nd+1$ for polynomials of degree at most d in each variable) and let $f : K^n \rightarrow K$ be a polynomial which is linear in each of its variables. Then $f(x)$ can be computed as follows. Uniformly at random, choose a line in K^n containing x. Query $n+1$ ($nd+1$ for polynomial of degree at most d in each variable) points on this line and interpolate the value of x.

The result is a random-self-reduction since each point queried has a uniform distribution over K^n by virtue of the fact that the line was chosen at random. (Of course, the $n+1$ queries together reveal that x is on this line, but this is not excluded from the definition.)

Arithmetization of boolean functions has been a standard technique to obtain random-self-reducible functions from arbitrary functions [1, 13, 14]. The technique consists of starting with a boolean function in n variables, and extending it to the unique multilinear polynomial in n variables over a field which coincides with the boolean function on all boolean inputs. This is the basis of the technique we use here to produce a function which is random-self-reducible.

2.3 Kolmogorov complexity

We use only the most simple notions from Kolmogorov complexity.

Definition 3 *The* **Kolmogorov complexity** *of a string x relative to a string y, written $C(x|y)$, is the length of the shortest program which on input y, produces the string x as output.*

We use the following fundamental fact about Kolmogorov complexity.

Proposition 2 *For any n and any string y there is a string x of length n such that $C(x|y) \geq n$.*

Such a string is called "incompressible" relative to y.

The reader is referred to the comprehensive textbook of Li and Vitányi [12] for more on Kolmogorov complexity.

3 Main result

Theorem 1 *There is a function which is random self-reducible by a two-round protocol, which is not coherent when nonadaptive queries are required, even if polynomial-length advice is allowed.*

In order to prove the theorem, we exhibit a function which has the former, but not the latter property. We begin by describing the function. In order to force two rounds of queries, the function is composed of three parts: two "hard" functions, and their composition. To successfully evaluate the function, the reduction must evaluate the composition in two steps. The difficulty is making this construction random-self-reducible, that is, allowing the process to be done in such a way that the queries do not reveal the input. This is where the multilinear extensions come in: by using this trick we can make random-looking, correlated queries that allow us to get the value of the function that we need without revealing the input we are interested in.

The first part of the proof, where we show that our function is random-self-reducible in two rounds, is straightforward, and no new tricks are introduced. The hard part, as usual, is the negative result: showing that the function is not coherent, even when we allow polynomial advice, if the queries must be made in a single round. This is where Kolmogorov complexity is used and new techniques are developed.

3.1 The counterexample

Our counterexample consists of a family of functions, each defined for inputs of a fixed length. Furthermore, the function is non-trivial (i.e., non-zero) only for "allowed" lengths which are (roughly speaking) towers of two. This is so that at any given length, when we consider a self-reduction, we may assume without loss of generality that queries made to smaller lengths are encoded in the advice string; and queries to bigger lengths are all zero since the next allowed length is too big to be queried by a polynomial-time algorithm. In what follows, we fix an allowed value for m and consider the function defined on strings of length slightly larger than m ($n = m + \log m$).

We start with two "hard" functions f and g, which we compose to obtain the third component of our final function. (It may be helpful to think of f as a treasure map, and g as a treasure function, and to get to the treasure, one must look up the map first.)

Fix m to be a tower of twos and define $f : \{0,1\}^m \to \{0,1\}^m$ and $g : \{0,1\}^m \to \{0,1\}^{m+1}$ as follows:

Choose a string σ of length $2^m(m + (m+1))$, incompressible relative to the string representing the integer m, that is, $C(\sigma|m) \geq 2^m(2m+1)$. Break σ into 2^m pieces of length m and 2^m pieces of length $m+1$. Let the first 2^m pieces represent the value of f on inputs of length m, and the rest represent the value of g in the same way.

We write $x' = (x.i)$ to mean the vector whose first m components are x and remaining components are the binary expansion of i.

Let $f'(x.i) = i$th bit of $f(x)$.

We take a multilinear extension of f and g as follows. Fix a field K of order $q = 2^{m+1}$ and let F be the multilinear extension of f' over K, G be the multilinear extension of g over K, where $f'(x)$ and $g(x)$ are interpreted as elements of K.

Putting the pieces together, we now define our function H_n (for $n = m + \log m$.) H_n has 3 parts: F, G, and $g(f(\cdot))$. Since the F part of the function is an extension of f', an index into the output of f is included in the input, and H_n is defined on vectors in K^n, $n = m + \log m$. The domain on the other parts of H is padded with zeros.

We define $H_n : K^n \times \{0,1,2\} \to K$. (This is to simplify notation and is equivalent to a function defined from strings of length $n\lceil \log |K| \rceil + 2$ to strings of length $\lceil \log |K| \rceil$.)

$$H_n(x.i, 0) = \begin{cases} g(f(x)) & \text{if } i = 0 \text{ and } x \in \{0,1\}^m \\ 0 & \text{otherwise.} \end{cases}$$
$$H_n(x.i, 1) = F(x.i)$$
$$H_n(x.i, 2) = \begin{cases} G(x) & \text{if } i = 0 \\ 0 & \text{otherwise.} \end{cases}$$

The construction is repeated at tower-of-two intervals (for m), and the function is zero everywhere else.

We argue in the two following sections that this function is random-self-reducible if adaptive queries are allowed, but if it is nonadaptively coherent with advice, then for some n "large enough", the string σ that gave rise to H_n can be compressed.

Since we focus our attention on a single large enough n in the course of the proof, the subscript n of H_n is omitted to simplify notation.

3.2 H is random-self-reducible

The argument for this part of the proof is a standard random-self-reducibility argument.

Claim 1 *H is random-self-reducible.*

Proof: It suffices to exhibit a random-self-reduction for H.

Protocol 0:

> **On input x':**
> If $x' = (x.0)$ and x in $\{0,1\}^m$:
> Compute $g(f(x))$ in two rounds: First use the standard random-self-reduction to compute $f(x)$ by making queries to the F part of H. Then compute $g(f(x))$ by making queries to the G part of H. Output $g(f(x))$
> Otherwise proceed as above with a randomly chosen x in $\{0,1\}^m$. Output 0.

Protocol 1:

> **On input x':**
> Using the standard random-self-reduction, compute $F(x')$ by making queries to the F part of H. Output $F(x')$.

Protocol 2:

> **On input x':**
> If $x' = (x.i)$ and $i = 0$:
> Using the standard random-self-reduction, compute $G(x)$ by making queries to the G part of H.
> Otherwise carry out the protocol for $x' = (x.0)$ as above, but output 0.

Protocol for H:

> **On input (x', j):**
> Simulate protocols 0,1,2 in parallel on input x'.
> Output the value returned by protocol j.

It is straightforward to see that this is a random-self-reduction for H. (The reason the 3 protocols are run in parallel is to hide the value of j.) □

3.3 H is not nonadaptively coherent with advice

To complete the proof of the main theorem, it remains to show that H is not coherent, even when polynomial advice is allowed, if the queries are to be made in a single round.

We will give a proof by contradiction. Assuming that the function is coherent, we will use the self-reduction to compress the description of the functions f and g that H was built up from. Since the string from which were obtained f and g was chosen to be incompressible, we get a contradiction.

In order to compute the value of the composition of f and g, the reduction has to obtain the value of f, and from that, make the relevant queries to the g part of the query to get the value of the composition of f and g. If the self-reduction manages to get the right value of the composition using a single round of queries, then it must already know (have a succinct encoding of) a part of f, or a part of g. By squeezing this information out of the reduction by simulating it, we end up with a more succinct description of f or of g, a contradiction. The main difficulty is to give a precise definition for what it means for a query to be "relevant." The details are given in the proof below.

Claim 2 *H is not nonadaptively coherent even with polynomial length advice.*

Assume A is an algorithm that witnesses the fact that H is nonadaptively coherent with polynomial length advice. We show that A can be used to compress the string that was used to define f and g. Let $a(n)$ be a polynomial bound on the length of the advice and $q(n)$ be a polynomial bound on the number of queries.

Idea of the proof: Consider the behavior of A on inputs $(x.0, 0)$ for x in $\{0,1\}^m$. A is allowed one round of queries to compute the value of $H(x.0, 0) = g(f(x))$. The functions f and g are random, so to query G on the correct value $(f(x))$ A must be tremendously lucky, or it must somehow already "know" f. On the other hand, if it doesn't query G to get its value on $f(x)$, then A must somehow already "know" g, since it will produce $g(f(x))$ as its output. A can therefore be made to either to produce information about f, or about g. This results in a compression of the functions.

In the case where A is lucky and makes the right queries to the G part of H, the queries themselves encode information about the value of $f(x)$, and the simulation need not be carried out further than the point where the queries are made. In the case where A's queries to G do not contain any information on $g(f(x))$, we are willing to provide A with the answers to its queries (which we know are irrelevant), and in exchange, A outputs "new" information about g, namely $g(f(x))$.

3.3.1 Working up to the algorithm

We may assume without loss of generality that A is deterministic. If not, then by parallel repetition we can raise the confidence level of A to the point where a single, polynomial length random string yields correct results on all inputs of length n. This string can be added to the advice string.

We start by introducing some notation.

Fix x in $\{0,1\}^m$. We will consider the behavior of A on inputs $(x.0,0)$, i.e., the $g(f(\cdot))$ part of H. We are interested in what queries are made which could yield relevant information about g. This includes queries to the G part of H, but also to the $g(f(\cdot))$ part, on inputs different from x. Let $Q(x) = q_1, ..., q_k$ be the set of these queries. More precisely, let $Q(x) = \{z | f(z') = z \text{ and } (z'.0,0) \text{ is queried on input } (x.0,0)\} \cup \{z | (z.0,2) \text{ is queried on input}(x.0,0)\}$. This notation can be extended to sets X in the natural way, i.e., for a subset $X \subseteq \{0,1\}^m, Q(X) = \bigcup_{x \in X} Q(x)$.

For any set of points S in K^m, let $D(S)$ be the set of points for which G is determined when the value of G on S is given.

Case 1: $f(x) \in D(Q(x))$. (In other words, A is lucky and makes a relevant query.) Then $f(x)$ can be described in $\log |D(Q(x)) \cap \{0,1\}^m|$ bits instead of m bits, by specifying only the index of $f(x)$ among $D(Q(x)) \cap \{0,1\}^m$.

Case 2: $f(x) \notin D(Q(x))$. Then $A(x)$ can be simulated with no information about $g(f(x))$. The output should yield full information about $g(f(x))$. (This is the general idea; there is a little more that needs to be done so that the details of the proof can go through.)

The following algorithm is used to reconstruct f and g. We start with an empty table with an entry for the value of f for each x in $\{0,1\}^m$ and one for the value of G for each x in K^m. As the algorithm progresses we fill up the table for f, then for G. The program assumes that the required information is provided as a stream of auxiliary input (this is tallied in the length of the program that reconstructs f and g.) Of course once G is fully determined, so is g.

In each iteration, an x in $\{0,1\}^m$ is chosen and A is simulated on x. If Case 1 applies to this x then A is only simulated up to the point where the queries are formulated. The value of $f(x)$ is deduced from the queries (used to compute $D(Q(x))$) and the index of $f(x)$ in $D(Q(x)) \cap \{0,1\}^m$, index which is provided on an auxiliary tape.

Case 2 requires a little additional trickery. If x is not in $D(Q(x))$, and $g(f(x))$ is as yet undetermined then it is tempting to say that by getting the answers to the queries and simulating A until it produces its output, namely $g(f(x))$, then we have discovered $g(f(x))$. Unfortunately it may be the case that the answers to these queries, *together with the information we have previously culled on* G determine $g(f(x))$. Then we cannot claim that A has generated "new" information; nor then has any compression taken place.

Let us introduce some more notation to deal with Case 2.

We call I the set of x (I for "independent") for which simulating A on $H(x.0,0)$ (the $g(f(\cdot))$ part) yields an as yet undetermined value, $g(f(x))$. We consider the elements of C_2 in lexicographical order, and add one element to I at each stage.

At stage 0, $I_0 = \phi$.
At stage $t > 0$:
  Let x_t be the smallest element of C_2 not yet considered such that $f(x_t) \notin D(Q(I_{t-1}))$.
  $I_t = I_{t-1} \cup \{x_t\}$.

Case 2.1 If I is small then $D(Q(I)) \cap \{0,1\}^m$ is small, and for each x in $C_2 \setminus I, f(x)$ must be in $D(Q(I \cup \{x\}))$. Therefore $f(x)$ has a short description as in Case 1.

Case 2.2 If I is large, then we may safely simulate A on all x in I and win on the description of $g(f(x))$.

The threshold between Cases 2.1 and 2.2 is $\frac{2^{m-1}}{mq(n)}$. Recall that $q(n)$ is an upper bound on the number of queries made on any given input of length n.

In order to complete the simulation of A on x (Case 2.2), we must be prepared to answer all queries to H. We have already dealt with queries made to instances on different lengths by defining our function only on "allowed lengths." Queries to the F part are handled easily since we will devise the algorithm so that we start out with a full description of f. Answers to queries to the G part of H are given on the auxiliary tape only if they are not already determined by the values already present in the table. Queries to the $g(f(\cdot))$ part of H can be handled like those to the G part, since f is known.

3.3.2 Compression algorithms

We are now ready to give the two algorithms to reconstruct f and g.

Algorithm 1 *when I is small ($\leq \frac{2^{m-1}}{mq(n)}$):*

For each x in $\{0,1\}^m \setminus I$
  {Cases 1 and 2.1; $f(x) \in D(Q(I \cup \{x\}))$}
  Simulate A on x to obtain $Q(x)$.
  Compute $D(Q(I \cup \{x\}))$.
  Read the index of $f(x)$ among $(Q(I \cup \{x\}))$ from the auxiliary tape.
  Find $f(x)$ and add it to the table for f.
Read the remaining values of f and all those of G from the auxiliary tape and add them to the table.

Algorithm 2 *when I is large ($> \frac{2^{m-1}}{mq(n)}$):*

> Read the values of f from the auxiliary tape and add them to the table for f.
> For each x in I
> {Case 2.2.}
> If $g(f(x))$ is undetermined:
> Read the answers to the queries off of the auxiliary tape,
> add them to the appropriate place in the table for g,
> compute the output and add it to the table for g as the value of $g(f(x))$.

3.3.3 Analysis

To complete the argument we must show that one of the programs above compresses the description of f and g. Instead of computing the length of each program, which would be messier, we compute the amount of compression obtained.

Let $\alpha = 2^m - |I|$ represent the number of inputs on which we can save by finding a value of f among the queries (Cases 1 and 2.2.)

For the first program: the price we pay to obtain each value of $f(x)$ is $\log |D(Q(I \cup \{x\})) \leq \log |I| + \log q(n)$, by Lemma 1 below. The profit we make is m bits. $|I|m$ bits suffice to encode the set I. The overhead cost for simulating A is $a(n)$, the length of the advice string, plus a constant c bits to encode the algorithm above and A's strategy.

Altogether, by using the first program, we obtain a compression of

$$\alpha(m - (\log |I| + \log q(n))) - |I|m - a(n) - c$$

bits. When $|I| \leq \frac{2^{m-1}}{mq(n)}$, this is at least

$$2^m(1 - \frac{m-1}{2mq(n)}) - a(n) - c$$

which is positive if m is large enough.

For the second program: For the x in I, at each iteration, we obtain $m+1$ bits of information. We may be unlucky and have two or more x in I map to the same value under f. However this cannot happen too often as we now argue. Let $\text{Im}_f(I)$ denote the image of I under f.

Claim 3 $|\text{Im}_f(I)| \geq \sqrt{|I|}$.

Proof: If $\text{Im}_f(I)$ is small, then the value of f on I can be described succinctly. To be precise, to describe f on I would require the following number of bits:

- To describe the image of I under f: $|\text{Im}_f(I)|m$ bits;
- To describe f on I: $|I|log(|\text{Im}_f(I)|)$ bits.

Since I is computable, and m is known, by this scheme we could save $|I|m - |\text{Im}_f(I)|m - |I|log(|\text{Im}_f(I)|)$ bits in the description of f. Since f is incompressible, this amount must not be greater than zero. If $|\text{Im}_f(I)| \leq \sqrt{|I|}$, the savings would be at least $|I|m - \sqrt{|I|}m - |I|log\sqrt{|I|}$, positive when m is large enough. Therefore we must have $|\text{Im}_f(I)| > \sqrt{|I|}$. $\square$

At each iteration at most $q(n)$ points in K^m are given away. By this observation and Lemma 2 below, the process can continue for at least $\sqrt{|I|}/q(n)$ iterations. Again, the overhead cost for the algorithm above, the advice and A is $a(n) + c$. We must also encode the set of $\frac{\sqrt{|I|}}{q(n)}$ points on which the algorithm is to be run, which requires an additional $m\frac{\sqrt{|I|}}{q(n)}$ bits.

By using the second program, we obtain a compression of

$$\frac{\sqrt{|I|}}{q(n)}(m+1) - \frac{\sqrt{|I|}}{q(n)}m - a(n) - c = \frac{\sqrt{|I|}}{q(n)} - a(n) - c$$

bits. When $|I| > \frac{2^{m-1}}{mq(n)}$, this is at least

$$\sqrt{\frac{2^{m-1}}{mq^3(n)}} - a(n) - c$$

which again is positive if m is large enough. $\square$

3.3.4 Algebraic lemmas

The only piece of the puzzle remaining are the proofs of Lemmas 1 and 2, which we give below.

Let K be a (finite or infinite) field, and Q a subset of K^n. Let us say that $x \in K^n$ is *determined* by $Q \subseteq K^n$ if any multilinear function $f : K^n \to K$ which is zero on Q also satisfies $f(x) = 0$. Let $D(Q)$ denote the set of such $x \in K^n$. (This is the same as saying that the values of g on Q determine $g(x)$ for any multilinear function $g : K^n \to K$.)

Lemma 1 $|D(Q) \cap \{0,1\}^n| \leq |Q|$.

Proof. Let M be the space of multilinear functions $f : K^n \to K$. This space has dimension 2^n (there are 2^n independent coefficients). Prescribing that $f(x) = 0$ for a given $x \in K^n$ means a homogeneous linear equation for these coefficients. The 2^n equations corresponding to $x \in \{0,1\}^n$ are linearly independent (since $f \in M$ is uniquely determined by its restriction to $\{0,1\}^n$ and this restriction is an arbitrary function).

Now let us consider the set $S(Q)$ of functions $f \in M$ which vanish on Q. This set is clearly a subspace: it is the solution space of a system of $|Q|$ homogeneous linear equations for the coefficients. Therefore the codimension of $S(Q)$ in M is at most $|Q|$. On the other hand, by the linear independence of the equations corresponding to the Boolean cube, we see that the codimension of $S(Q)$ is at least $|D(Q) \cap \{0,1\}^n|$. In summary: $|D(Q) \cap \{0,1\}^n| \leq codim S(Q) \leq |Q|$. $\Box$

Lemma 2 *Let $f : K^n \to K$ be a multilinear function. If the value of f is given on points $x_1, x_2, \ldots$ in such a way that for each i, $x_{i+1} \notin D(\{x_1, \ldots, x_i\})$, then the entire function is defined exactly when $i = 2^n$.*

Proof. Let F_i denote the left-hand side of the linear equation in the 2^n coefficients of f corresponding to prescribing the value of f at x_i. Then the condition says that F_{i+1} is not in the span of $\{F_1, \ldots, F_i\}$. Therefore the F_i are linearly independent, hence the process will terminate at $i = 2^n$ (the number of unknowns). $\Box$

We have shown that one of Algorithms 1 or 2 can always compress the description of f and g if the function H is coherent with polynomial advice. In either case, we obtain a contradiction since these were chosen to be incompressible. Therefore we have shown that H is not coherent with advice, thus concluding the proof of Theorem 1.

4 Open questions

There are several open problems related to this work. One is to apply these techniques to obtain separations in other contexts. Since random-self-reductions are at the center of cryptographic protocols, checkers, self-testers, correctors, probabilistically checkable proofs, and so on, these are natural candidates for further study.

Many problems remain to be solved in the case of random-self-reductions.

- Find a language which separates adaptive random-self-reducibility from nonadaptive coherent languages with advice. The proof above works just for functions.
- Find, under a reasonable complexity assumption, a set in **NP**$\backslash$**P** (or higher in the polynomial hierarchy) which separates adaptive random-self-reducibility from nonadaptive coherent functions with advice.
- Extend the current result to separate k-round random-self-reductions from $k+1$-rounds. Techniques developed in [11] could be useful here.

Acknowledgments

The second author would like to acknowledge the contributions of Lance Fortnow and Ashish Naik when we discussed this problem in 1996. We would like to thank Amber Settle and Barry Guiduli for several helpful discussions. Thanks also go to two anonymous referees for pointing out an error in the proof of Case 2.1.

References

[1] D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In *Proceedings, Symposium on Theoretical Aspects of Computer Science*, volume 415 of *Lecture Notes in Computer Science*, pages 37–48. Springer-Verlag, 1990.

[2] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Security with low communication overhead. In *Advances in Cryptology—CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 62–76. Springer-Verlag, 1991.

[3] R. Beigel and J. Feigenbaum. On being incoherent without being very hard. *Computational Complexity*, 2:1–17, 1992.

[4] H. Buhrman, L. Fortnow, D. van Melkebeek, and L. Torenvliet. Using autoreducibility to separate complexity classes. *SIAM Journal on Computing*, 1999. To appear.

[5] J. Feigenbaum. Locally random reductions in complexity theory. In *Advances in Computational Complexity Theory*, volume 13 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 73–98. American Mathematical Society, 1993.

[6] J. Feigenbaum and L. Fortnow. On the random-self-reducibility of complete sets. *SIAM Journal on Computing*, 22:994–1005, 1993.

[7] J. Feigenbaum, L. Fortnow, S. Laplante, and A. Naik. On coherence, random-self-reducibility and self-correction. *Computational Complexity*, 7:174–191, 1998.

[8] J. Feigenbaum, L. Fortnow, C. Lund, and D. Spielman. The power of adaptiveness and additional queries in random-self-reductions. *Computational Complexity*, 4:158–174, 1994.

[9] E. Hemaspaandra, A. Naik, M. Ogihara, and A. Selman. P-selective sets and reducing search to decision versus self-reducibility. *Journal of Computer and System Sciences*, 53(2):194–209, October 1996.

[10] R. Karp and R. Lipton. Some connections between nonuniform and uniform complexity classes. In *Conference Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, pages 302–309, 1980. An extended version has also appeared as: Turing machines that take advice, *L'Enseignement Mathématique*, 2nd series, 28:191–209, 1982.

[11] S. Laplante. *Kolmogorov Techniques in Computational Complexity Theory*. PhD thesis, University of Chicago Department of Computer Science, 1997.

[12] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Verlag, second edition, 1997.

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

[14] A. Shamir. IP=PSPACE. *Journal of the ACM*, 39(4):869–877, 1992.

[15] A. Yao. Coherent functions and program checkers. In *Proceedings of 22nd ACM Symposium on Theory of Computing*, pages 84–94, 1990.