RANDOM-SELF-REDUCIBILITY IN THE POLYNOMIAL HIERARCHY

CHUNG-CHIEH SHAN

ABSTRACT. In this paper, we define the concept of random-self-reducibility and provide intuitive motivation for the definition. We then introduce the concept of polynomial advice, and use it to prove the central result due to Feigenbaum and Fortnow: If some NP-complete language is random-self-reducible, then the polynomial hierarchy collapses at the third level.

1. INTRODUCTION

Intuitively, a function is said to be *random-self-reducible* if the evaluation of the function at each point can be reduced to the evaluation of the same function at one or more random points. Before giving the formal definition of random-self-reducibility, we first present its relation to RSA public key encryption and blind signatures.

The RSA public key encryption scheme [6, p. 467] works as follows. During key generation, Bob randomly selects three large numbers n, e and d, such that $ed \equiv 1 \pmod{\phi(n)}$. Here n is the product of two large primes, and ϕ is the Euler ϕ function; $\phi(n)$ is the number of integers between 1 and n that are relatively prime to n. From elementary number theory, for all x relatively prime to n we have

$$x \equiv x^{ed} \pmod{n}. \tag{1}$$

Bob publishes n and e while keeping d secret. All subsequent computation is done mod n. Let m be a message encoded as an integer mod n. Assume that m is relatively prime to n; because n is the product of two large primes, this assumption holds with high probability. To encrypt a message m, Alice computes the ciphertext $c = m^e$ and sends it to Bob. To decrypt c, Bob computes $c^d = m^{ed}$, which by (1) is equal to m.

One interesting property of the RSA encryption scheme is that decryption of any given message is as hard as decryption of random messages. More precisely, suppose that an eavesdropping adversary is able to decrypt a random message with probability p; in other words, among all $\phi(n)$ messages that Alice can send, the adversary is somehow able to decrypt a certain $p \cdot \phi(n)$ of them, without having been told the secret d. Now suppose that the adversary has some specific ciphertext that it wants to decrypt, say c. To decrypt c, the adversary randomly chooses some k such that k is relatively prime to n, and computes $c' = c \cdot k^e$. As long as k is uniformly randomly distributed among all integers mod n relatively prime to n, so are k^e and consequently c'. With probability p, then, the adversary can decrypt c' by somehow discovering m' such that $m'^e = c'$. This is equivalent to $m'^e = c \cdot k^e$, which implies $(m'/k)^e = c$; in other words, m'/k is the decryption of c.

Date: January 19, 1998.

The adversary can now decrypt c by computing $m = m'/k$. While this technique will only succeed with probability p, the adversary can simply keep trying different random values of k until it succeeds. Let N denote the expected number of attempts before the given message is decrypted successfully. We have

$$N = 1 + (1-p)N,$$

which gives $N = 1/p$. If p is a reasonably large value, say $p = 1/42$, then any given message can be decrypted by the adversary in about the amount of time it takes to decrypt 42 random messages. Intuitively, what is going on here is that the adversary is *reducing* a given instance of the RSA decrypted problem to a random instance. This means that even the worst case instance of RSA encryption is only as hard as the average case instance. We say that RSA decryption is 1-random-self-reducible.

The basic idea of RSA encryption can also be used to perform digital signatures [6, p. 41]. Suppose that Bob wants to publish a message m augmented with some kind of proof that m is indeed published by Bob. To do this, Bob computes the signature $s = m^d$ and publishes it along with m. Alice, having received both m and s, can then check the authenticity of m by computing s^e and comparing the result to m. If the signature is authentic, we have $s^e = m^{ed} = m$, again by (1).

On certain occasions, it may be desirable for Bob to perform a *blind signature* [6, p. 112]. In this scenario, Alice has a message m that she wants Bob to sign. Bob is willing to sign any message that Alice comes up with, but he does not want to disclose d, even to Alice. On the other hand, Alice does not want to disclose m, even to Bob. To make both sides happy, Alice first chooses a random *blinding factor* k, which must be a unit mod n. Alice computes $m' = m \cdot k^e$, then sends m' rather than m to Bob. Bob signs m' by computing $s' = m'^d$ and sending it back to Alice. Finally, Alice divides out the blinding factor by computing $s = s'/k$.

There are two things to note about this blind signature protocol. First, we can easily verify, again using (1), that $s = m^d$; in other words, the protocol works. Second, since k is uniformly randomly distributed, so are k^e and m'; the probability distribution of m' does not depends on m. Therefore, Bob has no idea what m is; in other words, the protocol works blindly. Using the blinding factor k, Alice has essentially reduced the specific problem instance of signing m to a random problem instance of signing m'. We say that RSA signing is 1-random-self-reducible.

In passing, we mention two real-world applications of blind signatures. The first application is digital postmarks. For purposes such as online intellectual property registration, the digital equivalent of postmarks is often needed. Say that Bob is a publicly trusted postmarking authority. Bob agrees to sign anything he receives before a certain time, then destroy d. The blind signature protocol above permits Bob to do his job without gaining any knowledge of trade secrets along the way.

The second application is digital cash [6, pp. 139–147]. One naïve way to implement a digital cash system is as follows. Suppose that Alice withdraws one dollar from her BobBank digital banking account. Bob signs the message "BobBank will pay one dollar upon presentation of this message", and sends the resulting "digital dollar bill" to Alice on a secure channel. Alice can now go to Carol's Candy Corner and make a payment using this dollar bill. Later, when Carol presents Alice's dollar bill to BobBank, BobBank checks the signature on the dollar bill and promptly deposits one dollar into Carol's account.

One obvious problem with this scheme is that, because every dollar bill carries the exact same message, the signature on every dollar bill is also identical. Thus, Alice can simply make copies of one single dollar bill and thereby create money out of thin air. To prevent this from happening, BobBank assigns a unique ID number to each dollar bill, so that each dollar bill carries a different message, and hence a different signature as well.

This solution, however, creates another issue, this time of privacy. Because Bob gets to generate all the unique ID numbers, he can keep track of exactly which dollar bills belong to whom. By correlating deposits with withdrawals, Bob can discover a list of people who shop at, say, Dave's Domination Dungeon. To fix this, we apply the blind signature protocol and let Alice, rather than Bob, create the dollar bill message. Bob blindly signs the message created by Alice without knowing the unique ID number on the dollar bill. This way, Bob has no way to correlate deposits with withdrawals.

It should be emphasized that this digression on digital cash is a gross oversimplification; more cryptography is needed to prevent many other possible exploits by the adversary.

2. Basic definitions

Let $\Sigma = \{0,1\}$ be the alphabet. Consider a function $f\colon \Sigma^* \to \Sigma^*$ which we would like to compute. We denote the input string in Σ^* by x, and let its length be $n = |x|$. In the following, f will often be the *characteristic function* of a language L; in other words, f will often be a function $\Sigma^* \to \Sigma$, representing the language

$$L = \{\, x \in \Sigma^* \mid f(x) = 1 \,\}$$

by mapping every string in L to 1 and every string not in L to 0.

Where random coin flips are involved, we use r to denote a sequence of random coin flips, uniformly distributed over $\Sigma^{w(n)}$. Here $w = |r|$ is the number of coin flips required; it is a polynomially bounded function of n.

Definition 1. Let $k\colon \mathbb{N} \to \mathbb{N}$ be a polynomially bounded function of n. The function f is said to be *nonadaptively k-random-self-reducible*, or *nonadaptively k-rsr*, iff there exist polynomial-time functions σ and ϕ satisfying the following two properties.

1. Correctness: For all n in $\mathbb{N}$ and x in Σ^n, the fraction of r's in $\Sigma^{w(n)}$ for which
$$f(x) = \phi\big(x, r, f(\sigma(1,x,r)), f(\sigma(2,x,r)), \dots, f(\sigma(k,x,r))\big)$$
holds is at least $3/4$.
2. Privacy: For all n in $\mathbb{N}$, all i between 1 and k, and all x_1, x_2 in Σ^n, if r is chosen uniformly from $\Sigma^{w(n)}$, then $\sigma(i, x_1, r)$ and $\sigma(i, x_2, r)$ are identically distributed random variables.

The function f is said to be *nonadaptively poly-rsr* iff there exists a polynomially bounded function k of n, such that f is nonadaptively k-rsr. A language L is said to be nonadaptively k-rsr or poly-rsr iff its characteristic function is nonadaptively k-rsr or poly-rsr, respectively.

Essentially, a function f is said to be nonadaptively k-random-self-reducible iff the evaluation of $f(x)$ can be reduced to the evaluation of f at k random points. The reduction is specified by the pair of functions σ and ϕ. The function σ takes the input x and the coin flip sequence r, and returns k new strings, namely $\sigma(1,x,r)$, …, $\sigma(k,x,r)$, for which the evaluation of f is requested. Once $f(\sigma(1,x,r))$, …, $f(\sigma(k,x,r))$ have somehow been evaluated, they are in turn fed to the function ϕ, which reverses the reduction and computes the final value of $f(x)$ with high probability.

For an intuitive picture, imagine that Alice wants to evaluate $f(x)$, and that Alice has k oracles for f at her disposal. Ordinarily, in order to find $f(x)$, Alice can simply pick any oracle and ask it for $f(x)$ directly. However, suppose that for some reason Alice does not want the oracles to know x. (Some of these reasons are mentioned in the introduction above.) Instead of asking for $f(x)$ directly, Alice asks the i-th oracle for $f(\sigma(i,x,r))$. She then applies the function ϕ to the oracle answers to compute $f(x)$. The correctness property in the definition above guarantees that Alice computes $f(x)$ correctly with high probability. The privacy property, on the other hand, guarantees that each oracle by itself has no information about x, except for its length n.

It should be emphasized that, while each oracle individually does not gain any information about x except for its length, there is no guarantee that multiple oracles cannot collaborate to gain more information about x than its length. In other words, in the privacy property above, although $\sigma(i,x_1,r)$ and $\sigma(i,x_2,r)$ are identically distributed for each individual i, the joint distribution of

$$\sigma(i_1,x_1,r),\sigma(i_2,x_1,r),\ldots,\sigma(i_m,x_1,r)$$

and the joint distribution of

$$\sigma(i_1,x_2,r),\sigma(i_2,x_2,r),\ldots,\sigma(i_m,x_2,r)$$

are not required to be identical when $m>1$.

As a digression on the subject of collaborating oracles, one can define *t-locally random reductions* as a generalization of nonadaptive random-self-reductions. With a locally random reduction, it is guaranteed that even if at most t oracles collaborate, they will not gain any information about x besides its length; in other words, the joint distribution of $\sigma(i,x,r)$ for at most t values of i depends only on the length of x. Nonadaptive rsrs correspond to the special case with $t=1$.

Definition 2. Let $k\colon \mathbb{N}\to\mathbb{N}$ be a polynomially bounded function of n. The function f is said to be *adaptively k-random-self-reducible*, or *adaptively k-rsr*, iff there exist polynomial-time functions σ and ϕ satisfying the following two properties.

1. Correctness: For all n in $\mathbb{N}$, x in Σ^n and r in $\Sigma^{w(n)}$, define the function γ recursively by

   $$\gamma(i,x,r)=\sigma\big(i,x,r,(f(\gamma(1,x,r)),f(\gamma(2,x,r)),\ldots,f(\gamma(i-1,x,r)))\big),$$

   where $i=1, 2, \ldots, k$. Then, the fraction of r's in $\Sigma^{w(n)}$ for which

   $$f(x)=\phi\big(x,r,\gamma(1,x,r),\gamma(2,x,r),\ldots,\gamma(k,x,r)\big)$$

   holds is at least $3/4$.
2. Privacy: For all n in $\mathbb{N}$, all i between 1 and k, and all x_1,x_2 in Σ^n, if r is chosen uniformly from $\Sigma^{w(n)}$, then $\gamma(i,x_1,r)$ and $\gamma(i,x_2,r)$ are identically distributed random variables.

The function f is said to be *adaptively poly-rsr* iff there exists a polynomially bounded function k of n, such that f is adaptively k-rsr. A language L is said to be adaptively k-rsr or poly-rsr iff its characteristic function is adaptively k-rsr or poly-rsr, respectively.

Intuitively, adaptive rsrs are rsrs in which answers from earlier oracle queries can be used to compute subsequent oracle queries. Regarding the privacy property, note that if incorrect answers are returned by the oracle in response to earlier queries, the distribution of later queries may leak information about x in addition to its length. In other words, the distribution of

$$\sigma\big(i, x, r, (y_1, \ldots, y_{i-1})\big)$$

is not required to depend only on n, unless $y_i = f(\gamma(i, x, r))$ for all i.

3. Polynomial advice

We next introduce the concept of *polynomial advice*. Intuitively, polynomial advice is additional information given to an algorithm, in addition to the usual input string. The advice string can depend only on the input length, not the actual contents of the input; furthermore, the length of the advice must be polynomially bounded in the input length. For example, the complexity class P/poly ("P with polynomial advice") can be defined as follows.

Definition 3. The complexity class P/poly is the set of languages recognizable in polynomial time with polynomial advice. More precisely, a language L is in P/poly iff there exists a function $v\colon \mathbb{N} \to \Sigma^*$, called the *advice*, and a polynomial-time "Arthur" function $f\colon \Sigma^* \times \Sigma^* \to \Sigma$, such that:

1. The length of $v(n)$ is polynomially bounded in n.
2. For all strings x, we have $x \in L$ iff $f\big(x, v(|x|)\big) = 1$.

Equivalently, as noted in [5, pp. 276–277], a language L is in P/poly iff for each n in $\mathbb{N}$ there exists a circuit C_n, such that the size of C_n is polynomially bounded in n, and C_n recognizes strings in L with length n.

In the above definition, note that v is not required to be even recursive; as a result, P/poly contains many languages that are not recursive either. For example, the language

$$\{\, x \mid \text{the length of } x \text{ encodes a terminating Turing machine} \,\}$$

is in P/poly, but is not recursive. On the other hand, because the advice v can only depend on the length of x, the additional information provided by the polynomial advice is also quite limited.

Similarly, we can define NP/poly ("NP with polynomial advice"), as well as $\mathrm{AM}^{\mathrm{poly}}$ and AM/poly (two flavors of "AM with polynomial advice").

Definition 4. The complexity class NP/poly is the set of languages whose proofs of membership are recognizable in polynomial time with polynomial advice. More precisely, a language L is in NP/poly iff there exists an advice function $v\colon \mathbb{N} \to \Sigma^*$, a polynomially bounded "Merlin length" function $u\colon \mathbb{N} \to \mathbb{N}$, and a polynomial-time "Arthur" function $f\colon \Sigma^* \times \Sigma^* \times \Sigma^* \to \Sigma$, such that:

1. The length of $v(n)$ is polynomially bounded in n.
2. For all strings x, we have $x \in L$ iff there exists some t such that $|t| < u(|x|)$ and $f\big(x, v(|x|), t\big) = 1$.

Equivalently, as noted in [5, pp. 276–277], a language L is in NP/poly iff for each n in $\mathbb{N}$ there exists a circuit C_n, such that the size of C_n is polynomially bounded in n, and C_n recognizes proofs of membership for strings in L with length n.

Definition 5. The complexity class $\mathrm{AM}^{\mathrm{poly}}$ is the set of languages for which there exists an Arthur-Merlin interactive proof protocol, with polynomial advice provided for Arthur's benefit. Given an input string x, Arthur randomly picks a coin flip sequence r, whose length is polynomially bounded in $|x|$; Merlin then replies with a proof that $x \in L$, which Arthur can check with the help of the polynomial advice. Formally, a language L is in $\mathrm{AM}^{\mathrm{poly}}$ iff there exists an advice function $v\colon \mathbb{N} \to \Sigma^*$, a polynomially bounded "Merlin length" function $u\colon \mathbb{N} \to \mathbb{N}$, a polynomially bounded "number of random bits" function $w\colon \mathbb{N} \to \mathbb{N}$, and a polynomial-time "Arthur" function $f\colon \Sigma^* \times \Sigma^* \times \Sigma^* \times \Sigma^* \to \Sigma$, such that:

1. The length of $v(n)$ is polynomially bounded in n.
2. Completeness: For all strings x and y, let $p(x, y)$ denote the fraction of r's in $\Sigma^{w(|x|)}$ for which there exists some t such that $|t| < u(|x|)$ and $f(x, y, r, t) = 1$. Then $p\big(x, v(|x|)\big) \geq 2/3$ for all x in L.
3. Soundness: Defining $p(x, y)$ as above, $p\big(x, v(|x|)\big) \leq 1/3$ for all x not in L.

The reason that the set defined above is denoted $\mathrm{AM}^{\mathrm{poly}}$ rather than AM/poly is that AM/poly is used to denote another flavor of AM with polynomial advice. While AM/poly is not actually used in this paper, we still define it below to point out a subtle issue, namely what happens when incorrect advice is given to Merlin.

Definition 6. The complexity class AM/poly is a subset of $\mathrm{AM}^{\mathrm{poly}}$, with the additional restriction that, even when given incorrect advice, the acceptance probabilities must still be either at least $2/3$ or at most $1/3$. More precisely, a language L is in AM/poly iff it is in $\mathrm{AM}^{\mathrm{poly}}$ according to the above definition, and for all strings x and v', either $p(x, v') \geq 2/3$ or $p(x, v') \leq 1/3$.

Given the multitude of definitions above, one might be disappointed to see the following result, which, as noted in [3], can be obtained by relativizing the main results of [1, 4].

Proposition 1. $\mathrm{AM}^{\mathrm{poly}} = \mathrm{AM/poly} = \mathrm{NP/poly}$.

Proof. By definition, $\mathrm{AM/poly} \subseteq \mathrm{AM}^{\mathrm{poly}}$. To see that $\mathrm{NP/poly} \subseteq \mathrm{AM/poly}$, simply note that Definition 6 becomes Definition 4 if the coin flip sequence r is ignored; in other words, NP/poly is simply the set of languages for which there exists a "Merlin proof protocol" with polynomial advice, and all Merlin proof protocols are Arthur-Merlin proof protocols as well.

It remains to be seen that $\mathrm{AM}^{\mathrm{poly}} \subseteq \mathrm{NP/poly}$. Let x be an input string and n be its length. By running the Arthur-Merlin protocol some polynomially bounded number of times in parallel and taking majority vote, we can improve the correctness probability of the protocol from at least $3/4$ to more than $1 - 2^{-n}$. Therefore, we can assume without loss of generality that the fraction of "bad" coin flip sequences (that is, the fraction of coin flip sequences that would cause Arthur to make a mistake on input x) is less than 2^{-n}. Since there are 2^n strings of length n, the fraction of coin flip sequences that are bad with respect to some input of length n is less than $2^{-n} \cdot 2^n = 1$. Hence, there exists at least one coin flip sequence that is not bad with respect to any input of length n. As long as Arthur sticks to this particular coin flip sequence, then, he would never err on any input of length n. To reduce from $\mathrm{AM}^{\mathrm{poly}}$ to NP/poly, simply use this coin flip sequence as the advice string for length n, in addition to whatever advice string there may already be for the $\mathrm{AM}^{\mathrm{poly}}$ protocol. $\square$

4. Random-self-reductions of NP languages

We are now ready to state and prove the main result of this paper, which is due to Feigenbaum and Fortnow [3].

Theorem 2. *If L is a language in* NP *and L is nonadaptively poly-rsr, then $\bar{L}$ (the complement of L) is in* $\mathrm{AM}^{\mathrm{poly}}$.

Let L be a nonadaptively poly-rsr language in NP, and let f denote its characteristic function; that is, define f by $f(x) = 1$ if $x \in L$, or 0 if $x \notin L$. We need to construct an $\mathrm{AM}^{\mathrm{poly}}$ interactive proof protocol for $\bar{L}$. A first attempt might be to follow the intuitive picture given below Definition 1. That is, Arthur randomly picks r, then reduces the single evaluation of $f(x)$ to the k evaluations of

$$f(\sigma(1, x, r)), \ldots, f(\sigma(k, x, r)).$$

As a shorthand, we let $x_i = \sigma(i, x, r)$ and denote the reduction process by

$$x \xrightarrow{r} x_1, \ldots, x_k.$$

We thus propose the following protocol.

Arthur: Randomly pick and send a coin flip sequence r of length $w(n)$.
Arthur and Merlin: Compute $x \xrightarrow{r} x_1, \ldots, x_k$.
Merlin: Compute and send $y_1 = f(x_1)$, …, $y_k = f(x_k)$.
Arthur: Compute $\phi(x, r, y_1, \ldots, y_k)$; accept (that $x \notin L$) iff it is 0.

The problem with this protocol, of course, is that Merlin can cheat by replying with incorrect values of $f(x_i)$. To prevent Merlin from cheating, there are two things that we need to worry about. First, we need to prevent Merlin from claiming that $f(x_i) = 1$ when in fact $f(x_i) = 0$; in other words, we need to prevent Merlin from claiming that $x_i \in L$ when in fact $x_i \notin L$. But this is easy, since L is in NP. All we need to do is to require Merlin to give a proof that $x_i \in L$, deterministically checkable in polynomial time, whenever he claims to Arthur that $f(x_i) = 1$.

On the other hand, we also need to prevent Merlin from claiming that $f(x_i) = 0$ when in fact $f(x_i) = 1$; in other words, we need to prevent Merlin from claiming that $x_i \notin L$ when in fact $x_i \in L$. Since we are not assuming that L is in coNP, we cannot simply require Merlin to produce a proof, as we did in the previous paragraph. Instead, note that by the definition of rsrs, the random distribution of x_i does not depend on x, as long as n is fixed. As a result, the distribution of $f(x_i)$ does not depend on x, either. In other words, the fraction of r's for which

$$\sigma(i, x, r) \in L$$

does not depend on x as long as n is fixed. Denote this fraction by

$$p_i(n) = \#\{\, r \in \Sigma^{w(n)} \mid \sigma(i, x, r) \in L \,\} \;/\; 2^{w(n)}. \tag{2}$$

Now suppose that, instead of picking one single coin flip sequence r, Arthur picks m coin flip sequences r_1, …, r_m. In the resulting reductions

$$\begin{aligned} x &\xrightarrow{r_1} x_{11}, \ldots, x_{k1}, \\ x &\xrightarrow{r_2} x_{12}, \ldots, x_{k2}, \\ &\;\vdots \\ x &\xrightarrow{r_m} x_{1m}, \ldots, x_{km}, \end{aligned}$$

each column on the right-hand side consists of m independently chosen strings from the same distribution. For each i between 1 and k, let Z_i denote the number of j's such that $x_{ij} \in L$, in other words, the random variable $\sum_{j=1}^{m} f(x_{ij})$. Statistically, then, we should have $Z_i \approx mp_i$ for sufficiently large m. Let us give $p_1, p_2, \ldots, p_k$ to Arthur as the polynomial advice. If Merlin claims values for $f(x_{ij})$ whose column-wise sums are too far off from the expected value mp_i, Arthur should decide not to accept. This way, if Merlin falsely claims that $x_{ij} \notin L$, Arthur would have a good chance of detecting it.

We now give a formal construction of the protocol and prove its correctness.

Proof of Theorem 2. Let L be a language in NP, and f be its characteristic function. Let σ, ϕ be a nonadaptive k-rsr for L. Although Definition 1 only requires the error probability of ϕ to be at most $1/4$, we can run the reduction a polynomially bounded number of times in parallel and take majority vote to lower the error probability to at most 2^{-n}.

For input strings x of length n, give $p_1(n)$, $p_2(n)$, $\ldots$, $p_k(n)$ as defined in (2) to Arthur as the polynomial advice. Let $m = 16k^3$. We claim that the following is an $\mathrm{AM}^{\mathrm{poly}}$ interactive proof protocol for $\bar{L}$.

Arthur: Randomly pick and send m coin flip sequences $r_l, \ldots, r_m$, each of length $w(n)$.
Arthur and Merlin: Compute $x \xrightarrow{r_j} x_{1j}, \ldots, x_{kj}$ for $j = 1, \ldots, m$.
Merlin: Compute and send $y_{ij} = f(x_{ij})$ for $i = 1, \ldots, k$, and $j = 1, \ldots, m$. Whenever $y_{ij} = 1$, send a proof that $x_{ij} \in L$.
Arthur: Accept (that $x \notin L$) iff the following three conditions hold:
1. For all $j = 1, \ldots, m$, the value $\phi(x, r_j, y_{1j}, \ldots, y_{kj})$ is 0.
2. For all i, j for which $y_{ij} = 1$, the proof sent by Merlin is correct.
3. For all $i = 1, \ldots, k$, we have $\sum_{j=1}^{m} y_{ij} > mp_i - 2\sqrt{km}$.

To prove the completeness of the protocol, suppose that x is indeed not in L, and that Merlin is honest. Because we have assumed without loss of generality that the error probability of ϕ is at most 2^{-n}, acceptance condition 1 can be met with probability at least $1 - m/2^n$, which is greater than $11/12$ for sufficiently large n, since $m = 16k^3$. Acceptance condition 2 can always be met. As for condition 3, note that for each i, the random variable Z_i has expected value $\mathrm{E}(Z_i) = mp_i$ and variance $\mathrm{Var}(Z_i) = mp_i(1 - p_i) < m$. By the Chebyshev inequality [2, p. 76],

$$\begin{aligned}
&\Pr\bigl(|Z_i - \mathrm{E}(Z_i)| \geq 2\sqrt{km}\bigr) \\
&\qquad \leq \Pr\bigl(|Z_i - \mathrm{E}(Z_i)| \geq 2\sqrt{k \operatorname{Var}(Z_i)}\bigr) \\
&\qquad \leq 1/4k.
\end{aligned} \tag{3}$$

Therefore,

$$\begin{aligned}
&\Pr\bigl(Z_i \leq mp_i - 2\sqrt{km}\bigr) \\
&\qquad \leq \Pr\bigl(|Z_i - mp_i| \geq 2\sqrt{km}\bigr) \\
&\qquad \leq 1/4k.
\end{aligned}$$

This shows that for each i, acceptance condition 3 is met with probability at least $1 - 1/4k$. Hence condition 3 is met for all i with probability at least $3/4$. In summary, all three acceptance conditions listed above can be met with probability at least $1 - 1/12 - 1/4 = 2/3$.

To prove the soundness of the protocol, suppose that x is in L, and Merlin tries to cheat. We need to prove that Arthur accepts with probability at most $1/3$. If Arthur accepts, then at least one of the following is true.

1. There exists a j, such that although Merlin sends honest replies $y_{ij} = f(x_{ij})$ to Arthur for all i, ϕ still makes a mistake and returns 0 erroneously.
2. For each j, Merlin lies for at least one x_{ij}, by sending $y_{ij} = 0$ when in fact $f(x_{ij}) = 1$.

The first case can happen with probability at most $m/2^n$, which is less than $1/12$ for sufficiently large n. We want to prove that the second case happens with probability at most $1/4$. The second case implies that Merlin tells at least m lies in total, which in turn implies that there is some column i in which Merlin tells at least m/k lies. It suffices to show that, for each i, Merlin can tell at least m/k lies in column i while still satisfying acceptance condition 3 with probability at most $1/4k$. For each i, this probability is simply the probability that at least $mp_i - 2\sqrt{km} + m/k$ of the strings x_{ij} in column i are in L. Note that $m/k = 16k^2 = 4\sqrt{km}$; again by (3) we have

$$\begin{aligned}\Pr\bigl(Z_i \geq mp_i - 2\sqrt{km} + m/k\bigr) \\ &\leq \Pr\bigl(|Z_i - mp_i| \geq 2\sqrt{km}\bigr) \\ &\leq 1/4k,\end{aligned}$$

as required. □

In order to relate random-self-reductions to the collapse of the polynomial hierarchy, we need the following result from [7].

Proposition 3. *If* coNP $\subseteq$ NP/poly*, then the polynomial hierarchy collapses at the third level.*

Proof. We will prove that coNP $\subseteq$ NP/poly implies $\Pi_3^{\mathrm{P}} \subseteq \Sigma_3^{\mathrm{P}}$. To prove this, let L' be any language in Π_3^{P}. Express L' in the form

$$x \in L' \iff \forall u_1 \exists u_2 \forall u_3\, P(x, u_1, u_2, u_3),$$

where P is some polynomial-time predicate, and all $\forall$ and $\exists$ quantifiers range over strings whose length are polynomially bounded in the length of x. We would like to translate L' into Σ_3^{P} form. By the hypothesis that coNP $\subseteq$ NP/poly, there exists some polynomial-time predicate Q and polynomial advice v such that

$$\forall u_3\, P(x, u_1, u_2, u_3) \iff \exists t\, Q(x, u_1, u_2, t, v)$$

holds for all x, u_1, and u_2. The desired translation is then

$$\begin{aligned}x \in L' &\iff \exists v' \,[\langle\text{the advice } v' \text{ works}\rangle \wedge \langle v' \text{ proves that } x \in L'\rangle],\\ \langle v' \text{ works}\rangle &\iff \forall u_1 \forall u_2 \forall y\, [\forall u_3\, P(y, u_1, u_2, u_3) \Leftrightarrow \exists t\, Q(y, u_1, u_2, t, v')]\\ &\iff \forall u_1 \forall u_2 \forall y\, \bigl[[\exists u_3\, \neg P(y, u_1, u_2, u_3) \vee \exists t\, Q(y, u_1, u_2, t, v')]\\ &\qquad\qquad \wedge [\forall t\, \neg Q(y, u_1, u_2, t, v') \vee \forall u_3\, P(y, u_1, u_2, u_3)]\bigr],\\ \langle v' \text{ proves } x \in L\rangle &\iff \forall u_1 \exists u_2 \exists t\, Q(x, u_1, u_2, t, v').\end{aligned}$$

It is easy to check that the above translation is indeed in Σ_3^{P}. □

We now have the following result, which suggests that NP-complete languages are not nonadaptively poly-rsr.

Corollary 4. *If there exists an* NP*-complete or* coNP*-complete language L that is nonadaptively poly-rsr, then the polynomial hierarchy collapses at the third level.*

Proof. The complement of any nonadaptively poly-rsr language is still nonadaptively poly-rsr. Therefore, we may assume that L is NP-complete. By Theorem 2, $\bar{L}$ is in $\mathrm{AM}^{\mathrm{poly}}$. By Proposition 1, $\bar{L}$ is in NP/poly. Because L is NP-complete, we have coNP $\subseteq$ NP/poly. Hence by Proposition 3, the polynomial hierarchy collapses at the third level. ☐

The results above generalize immediately to other complete sets in the polynomial hierarchy.

Corollary 5. *If there exists a Σ_n^{P}-complete or Π_n^{P}-complete language L that is nonadaptively poly-rsr, then the polynomial hierarchy collapses at the $(n+2)$-th level.*

Proof. The proofs of Proposition 1, Theorem 2, Proposition 3 and thus Corollary 4 relativize. We know that [5, pp. 425–426]

$$\Sigma_n^{\mathrm{P}} = \mathrm{NP}^{\Sigma_{n-1}^{\mathrm{P}}}, \quad \Pi_n^{\mathrm{P}} = \mathrm{coNP}^{\Sigma_{n-1}^{\mathrm{P}}}, \quad \Sigma_{n+2}^{\mathrm{P}} = \Sigma_3^{\Sigma_{n-1}^{\mathrm{P}}}, \quad \Pi_{n+2}^{\mathrm{P}} = \Pi_3^{\Sigma_{n-1}^{\mathrm{P}}}.$$

Therefore, simply relativize Corollary 4 with respect to an oracle for $\Sigma_{n-1}^{\mathrm{P}}$. ☐

Regarding adaptive reductions, the following result is proven in [3] by modifying the proofs of Theorem 2 and Corollary 4 above.

Theorem 6. *If L is a language in* NP *or* coNP *and L is adaptively $O(\log n)$-rsr, then $\bar{L}$ is in $\mathrm{AM}^{\mathrm{poly}}$. If there exists a Σ_n^{P}-complete or Π_n^{P}-complete language that is adaptively $O(\log n)$-rsr, then the polynomial hierarchy collapses at the $(n+2)$-th level.*

It is still an open problem whether an NP-complete language can be adaptively poly-rsr without resulting in the collapse of the polynomial hierarchy. In any case, the negative results above should not be taken to suggest that "hard" languages cannot be poly-rsr. Indeed, it is proven in [3] that PSPACE and EXP are nonadaptively poly-rsr; in an intuitive sense, languages need to be "powerful enough" in order to be capable of random-self-reduction.

References

[1] L. Babai and S. Moran, *Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes*, Journal of Computer and System Sciences, **36** (1988), 256–276.

[2] L. J. Bain and M. Engelhardt, *Introduction to probability and mathematical statistics*, 2nd ed., Duxbury, Belmont, 1991.

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

[4] S. Goldwasser and M. Sipser, *Private coins versus public coins in interactive proof systems*, in Proceedings of the 18th Annual ACM Symposium on Theory of Computing, Berkeley, 1986, 59–68.

[5] C. H. Papadimitriou, *Computational complexity*, Addison-Wesley, Reading, 1995.

[6] B. Schneier, *Applied cryptography: protocols, algorithms, and source code in C*, 2nd ed., Wiley, New York, 1996.

[7] C. Yap, *Some consequences of nonuniform conditions on uniform classes*, Theoretical Computer Science, **26** (1983), 287–300.