Low-end uniform hardness vs. randomness tradeoffs for AM

Ronen Shaltiel*
Department of Computer Science
University of Haifa
Mount Carmel, Haifa 31905, Israel.
ronen@haifa.ac.il

Christopher Umans†
Department of Computer Science
California Institute of Technology
Pasadena, CA 91125.
umans@cs.caltech.edu.

May 25, 2008

Abstract

In 1998, Impagliazzo and Wigderson [IW98] proved a hardness vs. randomness tradeoff for BPP in the *uniform setting*, which was subsequently extended to give optimal tradeoffs for the full range of possible hardness assumptions (in slightly weaker settings). In 2003, Gutfreund, Shaltiel and Ta-Shma [GSTS03] proved a uniform hardness vs. randomness tradeoff for AM, but that result only worked on the "high-end" of possible hardness assumptions.

In this work, we give uniform hardness vs. randomness tradeoffs for AM that are near-optimal for the full range of possible hardness assumptions. Following [GSTS03], we do this by constructing a hitting-set-generator (HSG) for AM with "resilient reconstruction." Our construction is a recursive variant of the Miltersen-Vinodchandran HSG [MV05], the only known HSG construction with this required property. The main new idea is to have the reconstruction procedure operate implicitly and locally on superpolynomially large objects, using tools from PCPs (low-degree testing, self-correction) together with a novel use of extractors that are built from Reed-Muller codes for a sort of locally-computable error-reduction.

As a consequence we obtain gap theorems for AM (and AM $\cap$ coAM) that state, roughly, that either AM (or AM $\cap$ coAM) protocols running in time $t(n)$ can simulate all of EXP ("Arthur-Merlin games are powerful"), or else all of AM (or AM $\cap$ coAM) can be simulated in nondeterministic time $s(n)$ ("Arthur-Merlin games can be derandomized"), for a near-optimal relationship between $t(n)$ and $s(n)$. As in [GSTS03], the case of AM $\cap$ coAM yields a particularly clean theorem that is of special interest due to the wide array of cryptographic and other problems that lie in this class.

1 Introduction

A fundamental question of complexity theory concerns the power of randomized algorithms: Is it true that every randomized algorithm can be simulated deterministically with small (say, subexponential) slowdown? Ideally, is a polynomial slowdown possible – i.e., is BPP $=$ P? The analogous question regarding the power of randomness in Arthur-Merlin protocols is: Is it true that every Arthur-Merlin protocol can be

*This research was supported by BSF grant 2004329 and ISF grant 686/07.

†This research was supported by NSF grant CCF-0346991, BSF grant 2004329, an Alfred P. Sloan Research Fellowship, and an Okawa Foundation research grant.

simulated by a nondeterministic machine with small slowdown? Is a polynomial slowdown possible – i.e., does AM $=$ NP? We refer to efforts to answer the first set of questions positively as "derandomizing BPP" and efforts to answer the second set of questions positively as "derandomizing AM". Recent work [IKW02, KI04] has shown that derandomizing BPP or AM entails proving certain circuit lower bounds that currently seem well beyond our reach.

1.1 The hardness versus randomness paradigm

An influential line of research initiated by [BM84, Yao82, NW94] tries to achieve derandomization *under the assumption* that certain hard functions exist, thus circumventing the need for proving circuit lower bounds. More precisely, we will work with hardness assumptions concerning the circuit complexity of functions computable in exponential time[1]. Derandomizing BPP can be done with lower bounds against size $s(\ell)$ deterministic circuits while derandomizing AM typically requires lower bounds against size $s(\ell)$ *nondeterministic* circuits, where ℓ is the input length of the hard function. Naturally, stronger assumptions – higher values of $s(\ell)$ – give stronger conclusions, i.e., more efficient derandomization. There are two extremes of this range of tradeoffs: In the "high end" of hardness assumptions one assumes hardness against circuits of very large size $s(\ell) = 2^{\Omega(\ell)}$ and can obtain "full derandomization," i.e., BPP $=$ P [IW97] or AM $=$ NP [MV05]. While in the "low-end" one assumes hardness against smaller circuits of size $s(\ell) = \mathrm{poly}(\ell)$ and can conclude "weak derandomization," i.e., simulations of BPP (resp. AM) that run in subexponential deterministic (resp. nondeterministic subexponential) time [BFNW93, SU05b]. Today, after a long line of research [NW94, BFNW93, Imp95, IW97, AK01, KvM02, MV05, ISW06, SU05b, Uma03, SU05a, Uma05] we have optimal hardness versus randomness tradeoffs for both BPP and AM that achieve "optimal parameters" in the *non-uniform* setting (see the discussion of non-uniform vs. uniform below).

1.2 Pseudorandom generators and hitting set generators

The known hardness versus randomness tradeoffs are all achieved by constructing a *pseudorandom generator* (PRG). This is a deterministic function G which on input m, produces a small set of T m-bit strings in time $\mathrm{poly}(T)$, with the property that a randomly chosen string from this set cannot be efficiently distinguished from a uniformly chosen m-bit string[2]. In this paper we are interested in a weaker variant of a pseudorandom generator called a *hitting set generator* (HSG). A function G is a HSG against a family of circuits on m variables, if any circuit in the family which accepts at least $1/3$ of its inputs also accepts one of the m-bit output strings of G (when run with input m). It is standard that given a HSG against deterministic (resp. co-nondeterministic) circuits of size $\mathrm{poly}(m)$ one can derandomize RP (resp. AM) in time $\mathrm{poly}(T)$ by simulating the algorithm (resp. protocol) on all strings output by the HSG, and accepting if at least one of the runs accepts. It is also known that HSGs against deterministic circuits suffice to derandomize two sided error (BPP) [ACR96, ACRT99].

The proofs of the aforementioned hardness versus randomness tradeoffs are all composed of two parts: first, they give an efficient way to generate a set of strings (the output of the PRG or HSG) when given access to some function f. Second, they give a *reduction* showing that if the intended derandomization using this set of strings fails, then the function f can be computed by a small circuit, which then contradicts the initial

[1] This type of assumption was introduced by [NW94] whereas the initial papers [BM84, Yao82] relied on cryptographic assumptions. In this paper we are interested in derandomizing AM which cannot be achieved by the "cryptographic" line of hardness versus randomness tradeoffs.

[2] An alternative formulation is to think of G as a function that takes a $t = \log T$ bit "seed" as input and outputs the element in T indexed by the seed.

hardness assumption when taking f to be the characteristic function of an EXP complete problem. We now focus on the reduction part. An easy first step is that an input x (to the randomized algorithm or AM protocol) on which the intended derandomization fails gives rise to a small circuit D_x that "catches" the generator, i.e., D_x accepts at least 1/3 of its inputs, but none of the strings in the generator output. (The obtained circuit D_x is a deterministic circuit when attempting to derandomize BPP and a co-nondeterministic circuit when attempting to derandomize AM). The main part of all the proofs is to then give a reduction that transforms this circuit D_x into a small circuit C that computes f.

1.3 Uniform hardness versus randomness tradeoffs

All the aforementioned hardness versus randomness tradeoffs are *nonuniform tradeoffs* because the reduction in the proof is nonuniform: given D_x it only shows the existence of a small circuit C that computes f, but doesn't give an efficient uniform procedure to produce it. (In other words, the reduction relies on nonuniform advice when transforming D_x into C). We remark that all the aforementioned results are "fully black-box" (meaning that they do not use any properties of the hard function f or circuit D_x) and it was shown in [TV02] that any hardness versus randomness tradeoff that is "fully black box" cannot have a uniform reduction.

A *non-black box* uniform reduction for derandomizing BPP in the low-end was given in [IW98]. This reduction gives a *uniform* randomized poly-time algorithm (sometimes called a *reconstruction algorithm*) for transforming a circuit D_x that catches the generator into a circuit C that computes the function f. It follows that if the intended derandomization fails, and if furthermore one can *feasibly generate* an input x on which it fails (by a uniform computation), then one can use the uniform reduction to construct the circuit C in probabilistic polynomial time, which in turn implies that f is computable in BPP. (This should be compared to the non-uniform setting in which one would get that f is in P$/poly$). An attractive feature of this result is that it can be interpreted as a (low-end) *gap theorem* for BPP that asserts the following: Either randomized algorithms are somewhat weak (in the sense that they can be simulated deterministically in subexponential time on feasibly generated inputs) or else they are very strong (in the sense that they can compute any function in EXP).[3] Obtaining a high-end version of this result is still open. In [TV02] it was shown how to get a high-end tradeoff in the slightly weaker setting where the hard function f is computable in polynomial space rather than exponential time.

1.4 Uniform hardness versus randomness tradeoffs for AM

A non-black-box uniform reduction for derandomizing AM in the high-end was given in [GSTS03]. It yields gap theorems for both AM and AM$\cap$coAM. The gap theorem for AM is analogous to that of [IW98] (except that it concerns the high-end and not the low end); it asserts: Either Arthur-Merlin protocols are very weak (in the sense that they can be simulated non-deterministically in polynomial time on feasibly generated inputs) or else they are somewhat strong (in the sense that they can simulate E $=$ DTIME$(2^{O(\ell)})$ in time $2^{o(\ell)}$). The gap theorem for AM $\cap$ coAM gives the same result with "AM" replaced by "AM $\cap$ coAM." The statement is in fact cleaner for AM$\cap$coAM because it does not mention feasibly generated inputs, and instead applies to *all inputs*.

The result of [GSTS03] relies on identifying a certain "resiliency property" of an HSG construction of [MV05] (constructed for the nonuniform setting) and on "instance checking" [BK95], which was previously used in this context by [BFL91, BFNW93, TV02]. While it gives a high-end result it does not generalize to

[3] To state this result formally one needs a precise definition of "feasibly generated inputs". The actual result also involves "infinitely often" quantifiers which we will ignore in this informal introduction.

the low-end because the HSG construction of [MV05] works only in the high end. We remark that there is an alternative construction (in the nonuniform setting) of [SU05b] that does work in the low-end but does not have the crucial resiliency property.

1.5 Our result: low-end uniform hardness versus randomness tradeoffs for AM

In this paper we obtain a resilient HSG (with a uniform reduction proving its correctness) that works over a larger domain of parameters and covers a wide range of hardness assumptions (coming very close to the absolute low-end). Using our result we extend the gap theorems of [GSTS03] as follows (for a formal statement of the two Theorems below see Theorems 2.4 and 2.5 in Section 2):

Theorem (informal) 1.1. *Either $E = DTIME(2^{O(\ell)})$ is computable by Arthur-Merlin protocols with time $s(\ell)$ or for any AM language L there is a nondeterministic machine M that runs in time exponential in ℓ and solves L correctly on* feasibly generated inputs *of length $n = s(\ell)^{\Theta\left(1/(\log \ell - \log\log s(\ell))^2\right)}$.*

The second Theorem below achieves a clean statement that works for all inputs (rather than feasibly generated inputs). However, this is only achieved for AM $\cap$ coAM.

Theorem (informal) 1.2. *Either $E = DTIME(2^{O(\ell)})$ is computable by Arthur-Merlin protocols with time $s(\ell)$ or for any AM $\cap$ coAM language L there is a nondeterministic (and co-nondeterministic) machine M that runs in time exponential in ℓ and solves L correctly on* all *inputs of length $n = s(\ell)^{\Theta\left(1/(\log \ell - \log\log s(\ell))^2\right)}$.*

Note that in the two theorems above we use a nonstandard way of measuring the running time of the machine M. This is because it is not possible to express the running time of M as a function of its input length in a closed form that covers all the possible choices of $s(\ell)$. It may be helpful to view the consequences for some particular choices of $s(\ell)$ and then express the running time of the nondeterministic machine as a function of the length of *its* input.

- For $s(\ell) = 2^{\Omega(\ell)}$ (the high-end) the nondeterministic machine runs in polynomial time in the length of its input. This is exactly the same behavior as in [GSTS03]. Thus, our results truly extend [GSTS03]. We comment that the techniques of [GSTS03] don't work when $s(\ell) < 2^{\sqrt{\ell}}$.
- For $s(\ell) = 2^{\ell^\delta}$ and constant $\delta > 0$, the nondeterministic machine runs in time $\exp((\log n)^{O(1/\delta)})$ on inputs of length n.
- For $s(\ell) = 2^{O(\log^a \ell)}$ and constant $a > 3$, the nondeterministic machine runs in time subexponential in the length of its input. The $a > 3$ requirement is suboptimal as we can hope to get the same behavior even when $a \geq 1$ (which is the absolute low-end).

A discussion regarding the best possible parameters that can be expected in hardness versus randomness tradeoffs appears in [ISW06]. Our results are suboptimal in the sense that one could hope to get $n = s(\ell)^{\Omega(1)}$ whereas we only get $n = s(\ell)^{\Omega(1/(\log \ell - \log\log s(\ell))^2)}$. Note that this is indeed optimal in the high-end, where $s(\ell) = 2^{\Omega(\ell)}$. However, it becomes suboptimal when $s(\ell)$ is smaller. Another effect of this problem is that while we can hope for hardness versus randomness tradeoffs that start working as soon as $s(\ell) = 2^{\omega(\log \ell)}$ (the "absolute low-end"), our results only start working when $s(\ell) > 2^{(\log \ell)^3}$.

1.6 Our techniques

The source of our improvement over [GSTS03] is that we replace the hitting set generator of [MV05] (that only works in the high-end) with a new construction of a generator. The new generator and its proof of correctness build on the previous construction of [MV05] while introducing several new ideas. We give a detailed informal overview of the ingredients and ideas that come up in our construction in Section 5.1.

On a very high level we can identify three new ideas in our construction. First, we use techniques from PCPs (low-degree testing and self-correction) to speed up certain steps in the reduction establishing the correctness of [MV05], so that they run in sublinear time in the size of their input. Although it has long been observed that there is some similarity between aspects of PCP constructions and aspects of PRG and HSG constructions, this seems to be the first time primitives like low-degree testing have proven useful in such constructions. Second, we run both the [MV05] construction and the associated reduction recursively, in a manner reminiscent of [ISW06, Uma05] (although the low level details are different). Finally, we introduce a new primitive called *local extractors for Reed-Muller codes*, which are extractors that are computable in sublinear time when run on inputs that are guaranteed to be Reed-Muller codewords. A construction of such an object can be deduced from [SU05b]. They play a crucial role in the improved constructions, and may be of interest in their own right. In Section 5.1 we give a detailed informal account of our construction and the way the new ideas fit into the proof.

1.7 Motivation

Uniform hardness vs. randomness tradeoffs represent some of the most involved proofs of non-trivial relationships amongst complexity classes, using "current technology." Pushing them to their limits gives new results, but also may expose useful new techniques, as we believe this work does. Moreover, the complexity classes we study, AM and AM$\cap$coAM, contain a rich array of important problems, from hard problems upon which cryptographic primitives are built, to group-theoretic problems, to graph isomorphism, and indeed all of the class SZK (Statistical Zero Knowledge).

A second motivation is the quest for *unconditional* derandomization results. In [GSTS03] it was shown that if one can prove a low-end gap theorem for AM that works for all inputs rather than just feasibly generated inputs, then it follows that AM can be derandomized (in a weak sense) *unconditionally* (the precise details appear in [GSTS03]). In this paper we come closer to achieving this goal by achieving a low-end version of [GSTS03].

1.8 Organization of the paper

In Section 2 we restate our main theorems formally using precise notation. In Section 3 we describe some ingredients that we use as well as the new "local extractors." In Section 4 we define some some new variants of AM protocols that we will use as sub-protocols in the final result. In Section 5 we give the new recursive HSG and an important ingredient that will be used in the proof. In Section 6 we state and prove the main technical theorem. In Section 7 we derive our main results from the main technical theorem.

2 Formal statement of results

In this section we formally state Theorems 1.1 and 1.2. In order to do so we need to precisely define the notion of "derandomization on feasibly generated inputs".

2.1 Feasibly generated inputs

Following [GSTS03] we will use the notions defined in [Kab01]. Loosely speaking, we say that two languages L, M are *indistinguishable* if it is hard to feasibly generate inputs on which they disagree. For this paper it makes sense to allow the procedure trying to come up with such inputs (which is called a *refuter* in the terminology of [Kab01]) to use nondeterminism. We first need the following definition.

Definition 2.1. *Let L_1, L_2 be two languages and let x be a string. We say that L_1 and L_2 disagree on x if $x \in (L_1 \setminus L_2) \cup (L_2 \setminus L_1)$.*

We now define the notion of a refuter, which is a machine attempting to distinguish between two languages.

Definition 2.2 (distinguishability of languages). *We say that a nondeterministic machine R (the* refuter*)* distinguishes *between two languages $L_1, L_2 \subseteq \{0,1\}^*$ on input length n if on every one of its accepting computation paths $R(1^n)$ outputs some x of length n on which L_1 and L_2 disagree.*

With this notation we can formally capture the informal statements in the introduction. More specifically, when given a language $L \in$ AM, a nondeterministic machine M running in time $t(n) < 2^n$ succeeds on feasibly generated inputs if for any refuter R running in time $t(n)$, R does not distinguish L from $L(M)$.[4]

2.2 Formal restatements of Theorems 1.1 and 1.2

We now restate our main theorems formally. We first require that the function $s(\ell)$ (which measures hardness) is a "nice" function in the standard way:

Definition 2.3 (time-constructible function). *A function $s(\ell)$ is* time-constructible *if:*

- *$s(\ell) \le s(\ell + 1)$, and*
- *$s(\ell)$ is computable from ℓ in time $O(s(\ell))$.*

The following Theorem is the formal restatement of Theorem 1.1. Note that we state the theorem for both classes E $= DTIME(2^{O(\ell)})$ and EXP $= DTIME(2^{\ell^{O(1)}})$ (the parameter choices for EXP are slightly different and appear in parenthesis). The statements below also use the notion of complete languages for E and EXP. Here we follow the standard convention and completeness for E is with respect to linear time reductions, whereas completeness for EXP is with respect to polynomial time reductions.

Theorem 2.4. *There exists a language A complete for E (resp. EXP) such that for every time-constructible function $\ell < s(\ell) < 2^\ell$, either:*

- *A has an Arthur-Merlin protocol running in time $s(\ell)$, or*
- *for any language $L \in$ AM there is a nondeterministic machine M that runs in time $2^{O(\ell)}$ (resp. $2^{\ell^{O(1)}}$) on inputs of length $n = s(\ell)^{\Theta\left(1/(\log \ell - \log\log s(\ell))^2\right)}$ (resp. $n = s(\ell)^{\Theta\left((1/\log \ell)^2\right)}$) such that for any refuter R running in time $s(\ell)$ when producing strings of length n there are infinitely many input lengths n on which R does not distinguish L from $L(M)$.*

[4] The statement in [GSTS03] uses a formal notation borrowed from [Kab01] that in the situation above reads AM $\subseteq$ [pseudo(NTIME($t(n)$))]–NTIME($t(n)$) where the first occurrence of NTIME($t(n)$) stands for the class of the refuter and the second one for the class of the machine M. We choose not to use this notation as it complicates the statements of our results and makes them less clear. However we stress that our results use exactly the same meaning of feasibly generated inputs as in [GSTS03, Kab01]. As in [GSTS03], this meaning of "feasibly generated" is incomparable with the one used in [IW98].

We remark that the hidden constants in the statement above depend on the language L. The following Theorem is the formal restatement of Theorem 1.2.

Theorem 2.5. *There exists a language A complete for E (resp. EXP) such that for every time-constructible function $\ell < s(\ell) < 2^\ell$, either:*

- *A has an Arthur-Merlin protocol running in time $s(\ell)$, or*
- *for any language $L \in AM \cap coAM$ there is a nondeterministic machine M that runs in time $2^{O(\ell)}$ (resp. $2^{\ell^{O(1)}}$) on inputs of length $n = s(\ell)^{\Theta\left(1/(\log \ell - \log\log s(\ell))^2\right)}$ (resp. $n = s(\ell)^{\Theta\left((1/\log \ell)^2\right)}$) such there are infinitely many input lengths n on which L and $L(M)$ are equal.*

Following [GSTS03] we can also reverse the order of "infinitely often" in Theorem 2.5 and achieve:

Theorem 2.6. *There exists a language A complete for E (resp. EXP) such that for every time-constructible function $\ell < s(\ell) < 2^\ell$, either:*

- *A has an Arthur-Merlin protocol running in time $s(\ell)$ which agrees with L on infinitely many inputs. (On other inputs the Arthur-Merlin protocol does not necessarily have a non-negligible gap between completeness and soundness), or*
- *for any language $L \in AM \cap coAM$ there is a nondeterministic machine M that runs in time $2^{O(\ell)}$ (resp. $2^{\ell^{O(1)}}$) on inputs of length $n = s(\ell)^{\Theta\left(1/(\log \ell - \log\log s(\ell))^2\right)}$ (resp. $n = s(\ell)^{\Theta\left((1/\log \ell)^2\right)}$) such that $L = L(M)$.*

3 Preliminaries

We assume that the reader is familiar with the definition of AM and other standard complexity definitions (see, e.g., [Gol98]). We remark that by [GS86, BM88, FGM$^+$89] we can assume that AM is defined by an Arthur-Merlin protocol with public coins, two rounds and perfect completeness. In this paper we also refer to protocols "that run in time $s(\ell)$ on inputs of length ℓ" by which we mean that the total length of messages sent during the protocol and the time of Arthur's final computation is bounded by $s(\ell)$.

3.1 Nondeterministic and co-nondeterministic circuits

We will be working with nondeterministic and co-nondeterministic circuits. A *nondeterministic circuit* is an ordinary Boolean circuit C with two sets of inputs, x and y. We say that C *accepts* input x if $\exists y\; C(x, y) = 1$ and that C *rejects* input x otherwise. We refer to a string y on which $C(x, y) = 1$ as *a witness* showing that C accepts x. A *co-nondeterministic circuit* has the opposite acceptance criterion: it *accepts* input x if $\forall y\; C(x, y) = 1$ and *rejects* input x otherwise. We refer to a string y on which $C(x, y) = 0$ as a *witness* that C rejects x.

3.2 Low degree testing and self correctors

The key to our results is that in many places we work *implicitly* with functions that are supposed to be low-degree polynomials – of course this is the central concept in PCPs as well. Just as with PCPs, we need the ability to locally test whether an implicitly supplied function is of the "correct" form: namely, we need to check whether it is (close to) a low-degree polynomial. As is standard, once we have determined that an implicitly supplied function is close to a low-degree one, we can "access" the nearby low-degree function locally using a self-corrector.

Low-degree testers and *self-correctors* are standard primitives in the PCP literature. In fact for our intended use of these primitives, we do not need delicate control of the parameters; we only need to be able to operate on d-variate functions over a field $\mathbb{F}$ in time $\mathrm{poly}(|\mathbb{F}|, d)$ (hence making at most that many queries), while handling constant relative distance, and with constant soundness error for both primitives. The formal definitions, and the known results that we will make use of follow:

Definition 3.1 (low-degree tester). *A* low-degree tester *with parameters h, δ, ϵ is a probabilistic oracle machine M that has oracle access to a function $f : \mathbb{F}^d \to \mathbb{F}$, and for which*

- *if $\deg(f) \leq h$ then M^f accepts with probability 1, and*
- *if every polynomial g with $\deg(g) \leq h$ satisfies $\Pr_x[f(x) \neq g(x)] \geq \epsilon$, then M^f rejects with probability at least δ.*

Lemma 3.2 ([FS95]). *There exists a (non-adaptive) low-degree tester with parameters $h, \delta, \epsilon = 2\delta$, running in poly$(|\mathbb{F}|, d)$ time, provided $|\mathbb{F}| > ch$, $\delta < \delta_0$, for universal constants c and δ_0.*

Definition 3.3 (self-corrector). *A* self-corrector *with parameters h, δ, ϵ is a probabilistic oracle machine M that has oracle access to a function $f : \mathbb{F}^d \to \mathbb{F}$, and for which*

- *if there exists a polynomial g of total degree h, for which $\Pr_x[g(x) \neq f(x)] < \epsilon$, then for all x*

$$\Pr[M^f(x) = g(x)] > 1 - \delta.$$

Lemma 3.4 ([BF90, Lip90]). *There exists a (non-adaptive) self-corrector with parameters $h, \delta = O(1/(\epsilon|\mathbb{F}|)), \epsilon$, running in poly$(|\mathbb{F}|, d)$ time, provided $\epsilon < \frac{1}{4}(1 - h/|\mathbb{F}|)$.*

We remark that for both low-degree testers and self-correctors, it is possible to decrease the soundness error from a constant to 2^{-t} by repeating the protocol $\Theta(t)$ times.

3.3 Local extractors for subsets

Another object we will use to perform local computations on an implicitly supplied function is what we call a "local extractor for subsets". The notion of "locally computable extractors" was introduced in [Lu04, Vad04] in the context of encryption in the bounded-storage model. Loosely speaking, it requires that the extractor is computable in time sublinear in the length of its first input. In our construction we require such extractors for very low "entropy thresholds". However, Vadhan [Vad04] proved that it is impossible to have such extractors unless the entropy threshold is very high. For this purpose we introduce a new variant of local extractors in which the first input comes from some prescribed subset (rather than the set $\{0,1\}^n$) and exploit the fact that we intend to run the extractor on inputs that are codewords in an error-correcting code. It turns out that the construction of [SU05b] can be computed in time polynomial in the output when applied on the Reed-Muller code, even when shooting for low entropy thresholds. The formal details follow:

Definition 3.5 (local extractor for subsets). *A (k, ϵ) local C-extractor is an oracle function $E : \{0,1\}^t \to \{0,1\}^m$ for which the following holds:*

- *for every random variable X distributed on C with minentropy*[5] *at least k, $E^X(U_t)$ is ϵ-close to uniform, and*
- *E runs in poly(m,t) time.*

Definition 3.6 (Reed-Muller code). *Given parameters r,h and a prime power q we define $RM_{r,h,q}$ to be the set of polynomials $p:\mathbb{F}^r \to \mathbb{F}$ over the field with q elements, $\mathbb{F}$, having degree at most h.*

The construction of [SU05b] gives the following local extractor for the Reed-Muller code (we have made no attempt to optimize the constants):

Lemma 3.7 (implicit in [SU05b]). *Fix parameters $r<h$, and let $C = RM_{r,h,q}$ be a Reed-Muller code. Set $k = h^5$. There is an explicit $(k, 1/k)$ local C-extractor E with seed length $t = O(r\log q)$ and output length $m = h = k^{1/5}$.*

The following proposition follows from the definition.

Proposition 3.8. *Let $E:\{0,1\}^t \to \{0,1\}^m$ be a (k,ϵ) local C-extractor, and let D be a subset of $\{0,1\}^m$. Then at most 2^k elements $x \in C$ satisfy: $\Pr_y[E^x(y) \in D] > \frac{|D|}{2^m} + \epsilon$.*

We will use local extractors in the following way. We will be interested in the set

$$\left\{x : \Pr_y[E^x(y) \in D] = 1\right\},$$

and we would like to be able to check whether some $x \in C$ is in this set by performing a local computation on x. This is not possible in general but a relaxation of this goal is. If we perform the probabilistic test of checking whether $E^x(y) \in D$ for a random y, then we will accept all x in the set, and not accept too many other x, because by the above proposition, the set of $x \in C$ on which this test accepts with high probability is "small" – it has size at most 2^k. This relaxation will turn out to be sufficient for our intended application.

4 The GST framework

In this section we describe the approach of [GSTS03] for obtaining uniform hardness vs. randomness tradeoffs for AM, and formalize two key ingredients, *commit-and-evaluate protocols* (which we define for the first time in this paper) and *instance checkers* (which have been defined in previous works).

As described in the introduction, our goal is to produce, from a hard function f, a HSG against nondeterministic circuits (and thus suitable for derandomizing AM) for which the associated reduction (showing how to compute f efficiently if the HSG fails) possesses an additional *resiliency* property. Here the *resiliency* property means that the reduction can be cast as two phases of interaction between Arthur and Merlin: in the first phase, Merlin commits (with high probability) to a function g, and in the second phase, Merlin reveals evaluations of g upon Arthur's request.

We formalize this two-phase protocol as a *commit-and-evaluate* protocol in the next subsection; the key properties it should posses are the aforementioned *resiliency*, meaning that Merlin is indeed committed to answering all future evaluation queries according to some function g after the commit phase, and *conformity with f*, which means that the honest Merlin can commit to the "true" hard function f. In Section 4.2,

[5]The minentropy of a random variable X is $\min_{x\in \text{supp}(X)} -\log(\Pr[X=x])$.

we define *instance checkers* which will be used to convert a commit-and-evaluate protocol that conforms resiliently with f into a true Arthur-Merlin protocol for f, assuming f is an E- or EXP-complete function.

Overall, we end up with a framework for obtaining uniform hardness vs. randomness tradeoffs for AM; the missing ingredient is a HSG whose associated reduction has the required resiliency property (i.e., can be formulated as a commit-and-evaluate protocol). This we construct in Section 5.

4.1 Commit-and-evaluate protocols

Let us start with some notation. An i round AM protocol is a protocol in which Arthur and Merlin receive a common input x and at each round Arthur sends public random coins and Merlin replies. At the end of the protocol Arthur outputs a value (not necessarily Boolean), denoted by $\text{out}(\pi, M, x)$, that is a random variable defined relative to a *strategy* M for Merlin; i.e., M is a function that describes Merlin's response given a history of the interaction so far. The value $\text{out}(\pi, M, x)$ is a random variable because Arthur flips coins during the protocol. The running time of the protocol is the running time of Arthur. A protocol may take an auxiliary common input y, which we will variously think of as a "commitment" or an "advice string". In this case we denote the output by $\text{out}(\pi, M, x, y)$. The output $\perp$ (which is intended to be output by Arthur when he detects a dishonest Merlin) is a distinguished symbol disjoint from the set of intended output values.

With this notation we can define the notion of AM protocols that output values:

Definition 4.1 (AM protocols that output values). *Given an AM protocol π and an input domain I, we say that π with auxiliary input y:*

- *is* PSV *(partially single valued) over I with soundness error s if there exists a function g defined over I, such that for all $x \in I$, and all Merlin strategies M^**

$$\Pr[\mathit{out}(\pi, M^*, x, y) \in \{g(x), \perp\}] \geq 1 - s.$$

- conforms *with a function f defined over I with completeness c if for all $x \in I$, there exists a Merlin strategy M for which*

$$\Pr[\mathit{out}(\pi, M, x, y) = f(x)] \geq c.$$

- computes *a function f over domain I with soundness error s and completeness c if π with auxiliary input y is PSV over I with soundness s and conforms with f with completeness c.*

We may sometimes omit s and c in which case they are fixed to their default values $s = 1/3$ and $c = 2/3$. We also omit I when it is clear from the context.

Note that a polynomial time AM protocol of the above type computes the characteristic function of some language L if and only if $L \in \text{AM} \cap \text{coAM}$. We will be interested in protocols that are composed of two phases, and operate over the domain $I = \{0,1\}^n$. The first phase is called the *commit phase*. This is an AM protocol whose input is 1^n, and whose auxiliary input is an advice string α that depends only on n. The role of this phase is to generate an auxiliary input to the second phase. The second phase is called the *evaluation phase*. This is an AM protocol whose input is $x \in I$, and whose auxiliary input is the output of the commit phase protocol. The reason we distinguish between two different phases is that we make the additional requirement that there is a function computed by the combined protocol and that this function is completely determined at the end of the commit phase (that is *before* Merlin knows the input x). The exact details appear below.

Definition 4.2 (commit-and-evaluate protocols). *A* commit-and-evaluate protocol *is a pair of AM protocols $\pi = (\pi_{commit}, \pi_{eval})$. Given π and an input domain $I = \{0,1\}^n$, we say that π with advice α:*

- conforms *with a function f defined over I if there exists a Merlin strategy M_{commit} for which*

$$\Pr[\pi_{eval} \textit{ with auxiliary input out}(\pi_{commit}, M_{commit}, 1^n, \alpha) \textit{ conforms with } f] = 1.$$

- *is* γ-resilient *over I if for all Merlin strategies M^*_{commit},*

$$\Pr[\pi_{eval} \textit{ with auxiliary input out}(\pi_{commit}, M^*_{commit}, 1^n, \alpha) \textit{ is PSV}] \geq \gamma.$$

- runs in time $t(n)$ *for some function t if both π_{commit} and π_{eval} run in time bounded by $t(n)$.*

We may sometimes omit γ, in which case it is fixed to its default value $\gamma = 2/3$.

We argue that completeness, soundness and resiliency of a commit-and-evaluate protocol can be amplified from their default values by parallel repetition.[6]

Proposition 4.3 (amplification of commit-and-evaluate protocols). *Let $\pi = (\pi_{commit}, \pi_{eval})$ be a commit-and-evaluate protocol that is resilient and conforms with f, with completeness 1, resiliency $2/3$ and soundness $1/3$. Furthermore, assume that π_{commit} is a two round protocol. Then π can be transformed (by parallel repetition) into a commit-and-evaluate protocol $\pi' = (\pi'_{commit}, \pi'_{eval})$ that is resilient and conforms with f, with completeness 1, resiliency $1 - 2^{-t}$ and soundness 2^{-t}. The transformation multiplies the running time and the output length of the commit protocol by $\Theta(t)$, and the running time of the evaluation protocol by $\Theta(t^2)$. The transformation preserves the number of rounds for both the commit protocol and the evaluation protocol.*

Proof. The new commitment protocol π'_{commit} simply runs the old commitment protocol π_{commit} $t' = \Theta(t)$ times in parallel, producing the commitments $(u_1, u_2, \ldots, u_{t'})$. Note that the Merlin strategy M'_{commit} that executes the honest M_{commit} strategy for each repetition results in *every* u_i being a commitment for which π_{eval} with auxiliary input u_i conforms with f with completeness 1. The new evaluation protocol π'_{eval} runs, for each i, the old evaluation protocol π_{eval} $t' = \Theta(t)$ times in parallel with u_i as auxiliary input. If for all commitments u_i all the repetitions of π_{eval} with advice u_i output the same value v then π'_{eval} outputs v and otherwise it outputs $\perp$.

Note that by the perfect completeness of π_{eval} if Merlin executes the "honest" M_{eval} strategy on each of the repetitions using advice u_i the resulting strategy causes Arthur to output $f(x)$ with probability one. Thus, π' conforms with f.

For resiliency note that as π_{commit} is a two round protocol then with probability at least $2/3$ over Arthur's choice of coins, every possible reply of Merlin results in a "good" commitment u (i.e., one for which π_{eval} is PSV). It follows that when making t' invocations of π_{commit}, with probability at least $1 - 2^{-t}$ there exists an i^* on which Arthur sends coin tosses for which every possible reply of Merlin leads to a "good" commitment u_{i^*}. We claim that whenever this event happens the protocol π'_{eval} is PSV when using advice $u_1, \ldots, u_{t'}$, which establishes the claimed resiliency.

[6] In the next proposition we only claim amplification for protocols where the commit protocol has two rounds and the evaluation protocol has perfect completeness. We make these relaxations because all protocols constructed in this paper have these properties. However, a more careful argument can get the same conclusion without these two assumptions. This follows along the same lines that parallel repetition of multi-round AM protocols amplifies soundness (see for example [Gol98, p.145–148]).

We have that for u_{i^*} there exists a function g such that for any Merlin strategy M_{eval},

$$\Pr[\text{out}(\pi_{\text{eval}}, M_{\text{eval}}, x, u_{i^*}) \in \{g(x), \perp\}] \geq 2/3.$$

It is folklore (see e.g. [Gol98, p.145–148]) that parallel repetition of (multi-round) AM protocols reduces the soundness error exponentially. Therefore, as π'_{eval} runs π_{eval} t' times with the commitment u_{i^*} it follows that any strategy of Merlin in π'_{eval} has probability at most 2^{-t} to output a value that is not in $\{g(x), \perp\}$ in all t' repetitions of π_{eval} with commitment u_{i^*}. In particular, no Merlin strategy for π'_{eval} can make Arthur output a value different than $g(x)$ with probability larger than 2^{-t}, which is the claimed soundness error. $\square$

4.1.1 Usefulness of commit and evaluate protocols

Note that after running the commitment protocol π_{commit} it is possible to run the evaluation protocol π_{eval} (with the auxiliary input that is output by π_{commit}) many times on many different inputs in I. We will typically perform these invocations of π_{eval} in parallel, and after suitably amplifying soundness (via Proposition 4.3), we can be sure that all evaluations agree with the committed-to function (with high probability). Note also that a γ-resilient commit-and-evaluate protocol that conforms with f does not necessarily "compute" f in any meaningful way. This is because in the commit phase, Merlin may not cooperate, causing the evaluation phase to receive an auxiliary input leading it to compute a function different from f. However, Merlin cannot choose this function in a way that depends on the input to the evaluation protocol.

On a more technical level, commit-and-evaluate protocol are useful because the commit phase can be executed *before* the input x is revealed, and following the commit phase it is guaranteed that Merlin is committed to *some* function f. This allows Arthur to make "local tests" on the function f. For concreteness let us demonstrate this approach on low-degree testing (that is, testing whether f is close to a low degree polynomial). Consider the following protocol: Arthur and Merlin play the commit phase of the protocol (which determines a function f). Then Arthur sends randomness for a low-degree test which in turn determines queries $x_1, \ldots, x_m$ to f. On each one of the queries x_i, Arthur and Merlin play the evaluation protocol (in parallel) and in the end Arthur checks that the low-degree test passes with the obtained evaluations. Note that no matter how Merlin plays he cannot make Arthur accept a function f that is far from a low degree polynomial. If Merlin was not required to commit to a function f in advance, he could answer queries arbitrarily, passing the low-degree test and then answering other queries (for example) in a manner inconsistent with any low-degree function.

4.2 Instance checkers

Following [GSTS03], it is possible to transform an AM protocol that *conforms resiliently* with an E-complete or EXP-complete function into one that *computes* the function. This is done by evaluating the function via an *instance checker* (defined below) following the commit phase. Thus to construct a (standard) AM protocol for languages in E or EXP it is sufficient to construct commit-and-evaluate protocols that conform resiliently with a complete problem.

Instance checkers were introduced in [BK95]. These are probabilistic oracle machines that are able to "check" that the oracle is some prescribed function in the sense that when given an "incorrect" oracle the machine will either fail or compute the prescribed function.

Definition 4.4 (instance checker). *Let $f : \{0,1\}^* \to \{0,1\}$ be a function. An instance checker IC for f with soundness error δ is a probabilistic oracle machine for which:*

- *for every $x \in \{0,1\}^*$, $\Pr[IC^f(x) = f(x)] = 1$, and*

- *for every function $g \neq f$ and every $x \in \{0,1\}^*$, $\Pr[IC^g(x) \in \{f(x), \perp\}] \geq 1 - \delta$.*

We say that an instance checker IC makes queries of length $v(\ell)$ on inputs of length ℓ if for every input $x \in \{0,1\}^\ell$ and for every oracle g all the queries made by IC to its oracle are for strings of length $v(\ell)$.

Note that by repeating the execution $\Theta(t)$ times the soundness error of instance checkers can be reduced from a constant to 2^{-t}. In this paper we use the fact that languages complete for EXP and E have instance checkers. This was achieved by a sequence of works [LFKN92, Sha92, BFL91, BFLS91]. The reader is referred to [TV02] for further details.

Theorem 4.5 (c.f. [TV02] Theorems 5.4 and 5.8). *The following hold:*

1. *There is a language in EXP that is complete for EXP under polynomial time reductions and its characteristic function f has a polynomial-time instance checker that makes queries of length $v(\ell) = \ell^{O(1)}$, where ℓ is the input length.*

2. *There is a language in E that is complete for E under linear time reductions and its characteristic function f has a polynomial-time instance checker that makes queries of length $v(\ell) = O(\ell)$, where ℓ is the input length.*

5 A recursive HSG construction

In this section we present a recursive version of the Miltersen-Vinodchandran (MV) generator [MV05], that receives a polynomial p (which should be thought of as the encoding of a hard function f) and outputs a multiset of m-bit strings.

5.1 Overview of the construction and proof

We start by describing the original MV generator using some of our language, and highlighting the parts that we modify to obtain improvements. The reader may skip to the formal, self-contained presentation of the new construction in Section 5.2 if they wish. Throughout this section, $\mathbb{F}$ is the field with q elements.

5.1.1 The original MV generator

Given a polynomial $p : \mathbb{F}^d \to \mathbb{F}$ of degree h the original MV generator chooses q and m to be slightly larger than h and (the standard choice is, say, $m = q = 2h$). For every axis-parallel line L in $\mathbb{F}^d$, it outputs the vector $z_L = (p_L(t))_{t \in \mathbb{F}}$ – the restriction of p to the line L.

Given a co-nondeterministic circuit D such that D rejects every output of the generator we would like to show that there is a commit-and-evaluate protocol π that receives D as advice and conforms with p resiliently. We need to make the additional assumption that D rejects very few – say 2^{m^δ} – strings of length m overall. In the context of AM derandomization this can be achieved by amplifying the AM protocol we are attempting to derandomize using dispersers. We stress, as this will be important later on, that this amplification can only achieve a constant $0 < \delta < 1$ efficiently.

We now describe the commit-and-evaluate protocol for evaluating p. In the commit phase Arthur sends a uniformly chosen set $S \subseteq \mathbb{F}$ of size $v \approx h^\delta$ and Merlin replies with a list of values for every point in S^d, that are supposed to be the "correct" set of values — the restriction of p to S^d. In the evaluation phase the two parties are given a point $x \in \mathbb{F}^d$ and Arthur wants to evaluate $p(x)$. Arthur and Merlin first compute a "path" to x: a sequence of axis-parallel lines starting with lines passing through S^d and proceeding through

$d-1$ sets of lines, each intersecting the previous set, until the final line which intersects x. This path has at most v^d lines, and for each line in the path, Merlin sends Arthur a univariate polynomial $g_L : \mathbb{F} \to \mathbb{F}$ (that is supposed to be the polynomial p restricted to L) by sending its $h+1$ coefficients. Arthur performs the following tests:

Small-set test Arthur asks Merlin to supply witnesses showing that D rejects g_L for all lines L in the path. (Note that Merlin can do this as D is a co-nondeterministic circuit).

Consistency test Arthur performs a "consistency test" using the polynomials g_L sent by Merlin. This consists of checking that for every pair of lines L_1 and L_2 that intersect at a point, the values of g_{L_1} and g_{L_2} agree at that point.

If both the tests pass then Arthur selects the single line L in the path that intersects x, and decides that $p(x)$ equals the value of g_L at that intersection, outputting that value.

We describe why this protocol conforms resiliently with p. An honest Merlin can indeed conform to p by following the protocol. A cheating Merlin has the freedom to choose values for the points in S^d that are incorrect and in this case the evaluation protocol does not necessarily conform with p. However the evaluation protocol is (with high probability over the choice of S) PSV by a key property of the consistency test: it is shown in [MV05] that that there is at most one collection of functions from the small set $Z = \{z : D \text{ rejects } z\}$ that passes the consistency test. This means that once Merlin commits to values for the points in S^d he cannot make Arthur output two different values on a given input x.

We stress that this key argument in [MV05] uses the structure of polynomials in a very weak way. The argument only uses that each g_L sent by Merlin is a set of $m > h$ evaluations of a degree h univariate polynomial and so it is a codeword of a Reed-Solomon code. In our construction we will use a relaxed notion of "lines" for which p restricted to such a "line" is still a codeword of an error-correcting code, which suffices for this argument. The precise definition of this relaxed "line" is in Definition 5.1.

We now turn our attention to the running time of the protocol. There are roughly v^d lines on the "path" to x and for each g_L Merlin needs to send $h+1$ coefficients to define g_L. Thus, overall the time is about hv^d. For the key property of the consistency test to hold, we need to set $v \approx m^\delta \approx h^\delta$ (this comes from the bound we have on the set Z, which in turn comes from the initial amplification of the AM protocol we are derandomizing). Overall the running time is about $h^{\delta d}$. Specifying the polynomial p explicitly requires roughly h^d coefficients and thus the protocol achieves something non-trivial since it runs in time that is only some constant root of h^d.

5.1.2 Goal: achieve the low end

The parameters achieved by the construction outlined above correspond to the "high-end" of hardness assumptions. When using this construction in the [GSTS03] framework, we will be given an E-complete language and set $f : \{0,1\}^\ell \to \{0,1\}$ be the characteristic function of this language (restricted to inputs of length ℓ). When given such a Boolean function f over ℓ bits we encode it as a $d = O(1)$ variate polynomial p (the low-degree-extension of f) with $h, m \approx 2^{\ell/d}$. We get that if we obtain a co-nondeterministic circuit D that rejects all outputs of the generator, then p (and therefore f and the complete language) have commit-and-evaluate protocols that conforms resiliently with p, and as described in Section 4, these can be transformed into AM protocols that compute p.

The overall protocol then gives us exactly the kind of parameters one wants; i.e., it runs in time polynomial in the output length, m, of the generator. However, this relationship is only achieved at the "high end", that is when $m = 2^{\Omega(\ell)}$, and in fact the construction fails completely when m becomes significantly smaller.

Our goal is to achieve the "low-end" so we must modify the construction of the generator so that we get a running time of poly(m) for any m, ideally all the way down to $m = \text{poly}(\ell)$.

5.1.3 Reducing the degree h and distinguishing between r, d

A very natural idea (that has been useful in previous works in this area, e.g., [STV01, SU05b]) is to encode the function f using a polynomial p with more than a constant number of variables. This will enable the encoding to use smaller degree. Note however that because the number of variables increases when the degree decreases, the running time of the protocol we just described does not benefit from reducing the degree h, as the gain over the trivial protocol depends only on δ which cannot be smaller than a constant. Thus, at this point it is not clear what we can gain from reducing the degree.

We will attempt to circumvent this problem by achieving the "best of both worlds": having a small degree while keeping the number of variables a constant. To achieve a behavior with that flavor we distinguish between two parameters r (the number of variables) and d (the number of "grouped variables"). More precisely, we now encode the function f as a polynomial $p : \mathbb{F}^r \to \mathbb{F}$ for super-constant r (at the absolute low-end we will use r as large as $\ell / \log \ell$ which allows the degree to go down to $h = \text{poly}(\ell)$). While doing so we keep d as a constant and identify $\mathbb{F}^r$ with B^d, where $B = \mathbb{F}^{r/d}$. This grouping, and the precise meaning of "line" in this generalized setting is stated in Definition 5.1.

We can now run the original MV generator just as before by thinking of p as a function $p : B^d \to \mathbb{F}$. This follows from our observation that we only need the restriction of p to the axis-parallel lines to form an error-correcting code, and here for every axis-parallel line L in B^d, the restriction of p to L (which is a function from $\mathbb{F}^{r/d}$ to $\mathbb{F}$) is a Reed-Muller codeword. In the commit-and-evaluate protocol for p that we already discussed, we only need to alter one thing to accommodate the grouped variables: when sending the functions g_L, Merlin will need to supply coefficients for p restricted to L which is now a degree h polynomial in r/d variables and has about $h^{r/d}$ coefficients (as compared to h coefficients previously).

At first glance it may seem that we have made progress and can handle m much smaller than the original MV construction required, but this is not the case. For the restrictions of p to the axis-parallel lines in B^d to form a code (which is needed for the key property of the consistency test), the HSG must output $m > h^{r/d}$ evaluations, and thus overall we do not gain (we were hoping to take m only slightly larger than h, not $h^{r/d}$.). However, we did make some progress as various quantities in the protocol (such as the number of evaluations Merlin must supply in the commitment phase, and the length of the "path" computed for each evaluation) depend on d (which is constant) rather than on r.

5.1.4 Reducing m by using local extractors for Reed-Muller codes

We will reduce m by modifying the generator construction further. For each axis-parallel line L in B^d, instead of outputting enough evaluations of p restricted to L to induce an error-correcting code, we will use an extractor. More precisely, we take E to be an extractor with output length $m \approx h$, and we output the strings $E(p_L, y)$ for all possible seeds y, and p_L ranging over all restrictions of p to axis-parallel lines L in B^d.

Then, in the commit-and-evaluate protocol, we can replace the small-set test with a *probabilistic* small-set test: given the polynomial g_L sent by Merlin (which is supposed to be p restricted to L) we check that D rejects $E(g_L, y)$, for a random y. All of the polynomials g_L that formerly passed the small-set test will still do so, since by assumption all of the outputs of the generator (and thus all of the outputs of E run on restrictions of p to axis-parallel lines in B^d) are rejected by D. At the same time, by the extractor property, there can be only a small number (say, 2^{m^2}) of strings that pass the new probabilistic test with reasonable probability. This will be sufficient to maintain the resiliency of the protocol.

However, our goal was to reduce m and have the protocol run in time poly(m). But even invoking the extractor once for the probabilistic small-set test takes time linear in its input length $h^{r/d}$, which is much larger than m.

The crucial realization at this point is that we are only ever interested in running the extractor on input strings that are evaluations of low-degree polynomials! We can thus replace E with a local extractor for the Reed-Muller code, and consequently reduce the running time of the extractor to poly(m) when given oracle access to its input.

So, we can perform the small-set test in time poly(m), given oracle access to the various g_L sent by Merlin. For our choice of parameters, the consistency test will also run in time poly(m) given oracle access to the g_L. However one hurdle remains: the step in which Merlin sends the coefficients of the polynomials g_L still requires $h^{r/d} \gg m$ time to send the $h^{r/d}$ coefficients of g_L, while we are shooting for poly(m) time.

5.1.5 Sending the polynomials g_L implicitly

Let us assume at this point that for some reason we already knew that for every axis-parallel line L in B^d, the polynomial p restricted to L has a commit-and-evaluate protocol that conforms with it resiliently and that this protocol runs in time poly(m). Then instead of having Merlin send the polynomial g_L (which is supposed to be p restricted to L) explicitly, Arthur and Merlin could play the commitment phase of the protocol for g_L, after which Merlin will be able to assist Arthur in evaluating g_L on any input that Arthur wishes.

However, we have now exposed the protocol to the possibility that Merlin may cheat by committing to a function that is not a low-degree polynomial, and then (at least) two things break: the local extractor for Reed-Muller codes may be run with access to an oracle that is not a Reed-Muller codeword, destroying the extractor property needed for the integrity of the small-set test; and, the key property of the consistency test may fail, as it relies on all of the received functions being codewords.

The solution is to run a low-degree test on each function Merlin commits to, verifying that it is indeed a low-degree polynomial. This test can be done locally, with oracle access to the function, and the fact that Merlin is *committed* to a function (and cannot alter the requested evaluations upon seeing the randomness of the test) ensures the validity of the test.

Let us summarize our current position. *If* we knew that for every axis-parallel line L in B^d the polynomial p restricted to L had a poly(m) time commit-and-evaluate protocol that conformed with it resiliently, then we would be able to produce a commit-and-evaluate protocol that conforms with p resiliently, and more importantly, runs in time poly(m) (which is our goal).

5.1.6 Using recursion to obtain commit-and-evaluate protocols for the polynomials g_L

It is important to note that when trying to construct a protocol for a polynomial p with r variables, we need to assume the existence of a protocol for polynomials g_L with a smaller number, r/d, of variables. This will allow us to use recursion. The base case will be the original MV generator, where $r = d$. For the base case we already showed how to construct a commit-and-evaluate protocol that runs in time poly(m).

To give us the commit-and-evaluate protocol on the restrictions of p to axis-parallel lines L in B^d, needed in the recursive step, we modify the construction of the original HSG, finally arriving at the construction in Figure 1. In this construction, in addition to the original output of our modified MV generator run on p, we also output all the outputs of our modified MV generator run on the polynomials p restricted to each axis-parallel line L in B^d, and continue with this recursively. The inputs to the recursive calls are sufficiently smaller than the original input so that we do not increase the set of outputs of the generator by more than a polynomial factor. Now, a circuit D that rejects all the outputs of our generator can be used as advice to play the protocol on all the polynomials g_L that we will ever be interested in at any level of the recursion.

Whenever the final commit-and-evaluate protocol needs to access some restriction of p to a line L, it will invoke the protocol now available for that restriction, continuing this recursively down to the base case.

We stress that the resiliency property of the commit-and-evaluate protocols plays a crucial role inside the recursion (in addition to its role as described in Section 4). Specifically, the resiliency property of the protocol for g_L says that following the commitment phase, Merlin is committed to some function, and this is what prevents Merlin from cheating when doing the local tests (such as the low degree test). If it wasn't for resiliency then Merlin would be able to choose outputs for g_L *after* seeing the queries of the low degree test which would make the test useless.

5.1.7 Losses suffered in the recursion

While we can reduce m using the ideas outlined above, there are also some costs to using this recursive argument. First, each recursive step in the protocol picks up two additional rounds and thus we end up with a protocol with $2\log_d r$ rounds. Such protocols can be transformed into two round protocols but the running time suffers a blowup which is slightly super-polynomial. The running time also suffers as each recursive step multiples the running time of the protocol by $\text{poly}(m)$. When taking these two factors into consideration and transforming to a two round AM protocol we get that this protocol has running time $m^{O(\log_d^2 r)}$ rather than $m^{O(1)}$. This accounts for the slight non-optimality of our main gap theorems.

5.2 The construction

We now give the full recursive HSG construction, which uses the following definition. Recall that $\mathbb{F}$ is the field with q elements.

Definition 5.1 (grouping variables and MV lines). *Given a function $p : \mathbb{F}^r \to \mathbb{F}$ and a parameter d that divides r we define $B = \mathbb{F}^{r/d}$ and identify p with a function from B^d to $\mathbb{F}$.*

Given a point $x \in B^d$ and $i \in [d]$ we define the line passing through x in direction i *to be the function $L : B \to B^d$ given by $L(z) = (x_1, \ldots, x_{i-1}, z, x_{i+1}, \ldots, x_d)$. This is an axis-parallel, combinatorial line, which we call an* MV line *for short.*

Given a function $p : \mathbb{F}^r \to \mathbb{F}$ and an MV line L we define a function $p_L : B \to \mathbb{F}$ by $p_L(z) = p(L(z))$.

Note that if $p : \mathbb{F}^r \to \mathbb{F}$ is a polynomial then $p_L : \mathbb{F}^{r/d} \to \mathbb{F}$ is also a polynomial with degree bounded by that of p. We present our construction in Figure 1.

Lemma 5.2. *The construction $RMV_{h,d}(p)$ runs in time $q^{O(r)}$ and outputs at most $q^{O(r)}$ strings.*

Proof. Let $r = d^i$. We show by induction on i that the running time and number of output strings is bounded by q^{cr} for some universal constant c.

For the base case, when $r = d^1$, the number of MV lines is at most q^r, and the number of output strings produced from each line is at most $q^{c'r/d}$ for some universal constant c'. We are using the fact that the specified local C-extractor has at most $\text{poly}(q^{r/d})$ seeds, where $q^{r/d}$ is the blocklength of code C. In total the running time and the number of strings is at most $q^{r+c'r/d} \le q^{cr}$.

In general, when $r = d^i$, the number of MV lines is at most q^r and for each line, we produce $q^{c'r/d}$ strings. By induction the recursive call generated for each line has running time and number of strings

Input A multivariate polynomial $p : \mathbb{F}_q^r \to \mathbb{F}_q$ of degree h.

Output A multiset of m bit strings.

Parameters and requirements We require that r is a power of d and that h is a prime power. We set $q = h^{100}$ and $m = h^{1/100}$.

Ingredients The $(k, 1/k)$ local C-extractor E from Lemma 3.7 for Reed Muller code $C = \mathrm{RM}_{r/d,h,q}$. Note that $k = h^5$, the extractor uses seed length $O((r/d) \cdot \log q)$ and (by using only a prefix of the output) it outputs m bits.

Operation of RMV$_{h,d}(p)$:

- Set $B = \mathbb{F}_q^{r/d}$. For every $x \in B^d$ and $i \in [d]$, let $L : B \to B^d$ be the MV-line passing through x in direction i. Note that p_L is an element of the Reed-Muller code $\mathrm{RM}_{r/d,h,q}$. Compute $E^{p_L}(y)$ for all seeds y. Let H_p denote the set of these m bit strings, as L ranges over all MV lines.
- If $r = d$ then output H_p.
- If $r > d$ then for each MV line L make a recursive call to $\mathrm{RMV}_{h,d}(p_L)$. Note that while the dimension of p was r, the dimension of p_L is r/d. Each one of these recursive calls returns a multiset of m bit strings that we will call H_L. Output the union of H_p and H_L as L ranges over all MV lines.

Figure 1: Recursive MV generator $\mathrm{RMV}_{h,d}(p)$

bounded above by $q^{cr/d}$. So we have an overall bound of $q^{r+c'r/d} + q^{r+cr/d}$, which is less than q^{cr} for a suitably chosen universal constant c. □

5.3 Miltersen-Vinodchandran consistency test

We abstract a certain part of the original Miltersen-Vinodchandran proof [MV05], and prove conformity and resiliency for it. This primitive, together with the three primitives in Sections 3.2 and 3.3 will be the main ingredients in the reduction proving correctness of the new generator. The main point of the abstraction is that the consistency test makes sense when the "lines" of the original MV construction are replaced by what we are calling "MV lines," which are more general. We need one definition first:

Definition 5.3 (MV paths and S-boxes). *Given $x \in B^d$ and a set $S \subseteq B$ we define a sequence of d sets $T_1, \ldots, T_d$ called the* MV path to x using S. *Each of these sets contains MV lines as follows: T_i contains all MV lines through points $\{(x_1, \ldots, x_i, s_{i+1}, \ldots, s_d) : s_{i+1}, \ldots, s_d \in S\}$ in direction i. We say that a line L appears in the MV path if $L \in \cup_i T_i$. Given a set $S \subseteq B$, an S-*box *is a function $a : S^d \to \mathbb{F}$.*

Note that in the above definition, for $|S| > 1$, there are $\Sigma_{i=1}^d |S|^{i-1} \leq |S|^d$ MV lines appearing in the MV path.

Figure 2 describes a test that we call the "MV consistency test". The usefulness of this procedure is captured in the following lemmas:

Input A point $x \in B^d$, a subset $S \subseteq B$, and an S-box $a : S^d \to \mathbb{F}$. Also, the following collection of functions: for every line L appearing in the MV path to x using S, a function $g_L : B \to \mathbb{F}$.

Operation Let $T_1, \ldots, T_d$ be the MV path to x using S. The MV consistency test passes if the two tests below pass:

- (agreement with the S-box) For every line L in T_1 and $z \in S$, we check that $g_L(z) = a(L(z))$.
- (agreement at intersection points) For all $i = 2, 3, \ldots, d$, for every pair of lines $L_1 \in T_{i-1}$ and $L_2 \in T_i$: if $L_1(z_1) = L_2(z_2)$ for some z_1, z_2, we check that $g_{L_1}(z_1) = g_{L_2}(z_2)$.

Figure 2: MV consistency test

Lemma 5.4 (conformity of MV consistency test). *Fix a function $p : B^d \to \mathbb{F}$, an $x \in B^d$, and a subset $S \subseteq B$. The MV consistency test passes when given as input x, S, the S-box $a : S^d \to \mathbb{F}$ defined by $a(s_1, \ldots, s_d) = p(s_1, \ldots, s_d)$, and the collection of functions p_L ranging over all MV lines L in the MV path. Furthermore, if L is the single line in T_d, then $p_L(x_d) = p(x)$.*

Proof. Since all of the functions p_L and the S-box a agree with a single, underlying function p, it is clear that these inputs pass the MV consistency test. The second item follows from the definition of T_d. ☐

Lemma 5.5 (resilience of MV consistency test). *Let Z be a set of at most K functions where each one is a function from B to $\mathbb{F}$ and assume that for any two functions $g_1, g_2 \in Z$, with $g_1 \neq g_2$,*

$$\Pr_{z \in B}[g_1(z) = g_2(z)] \leq \beta.$$

Then with probability at least γ over the choice of a random subset $S \subseteq B$ with

$$|S| \geq (2 \log K + \log(1/(1-\gamma)))/\log(1/\beta)$$

the following event holds: for every S-box $a : S^d \to \mathbb{F}$ and for every x, there is at most one collection of functions from Z that passes the MV consistency test.

Proof. Let us call a subset $S \subseteq B$ of the specified size "good" if it separates the functions $g \in Z$; i.e., for all $g_1 \neq g_2$, there is some $s \in S$ for which $g_1(s) \neq g_2(s)$. It is a standard calculation to see that the probability a randomly chosen S of the specified size is not "good" is at most $K^2 \beta^{|S|}$, which is at most $1 - \gamma$ by our choice of $|S|$.

Now fix an S-box a and some x. Let $T_1, T_2, \ldots, T_d$ be the MV path to x using S. By the definition of "good," for each MV line $L \in T_1$, there is at most one function $g_L \in Z$ satisfying $g_L(s) = a(L(s))$ for all $s \in S$.

The crucial observation is that for each MV line $L_2 \in T_2$, the union of the intersections of L_2 with the MV lines in T_1 is exactly $L_2(S)$. Therefore (again using the definition of "good") for each $L_2 \in T_2$, there is at most one function $g_{L_2} \in Z$ for which $g_{L_2}(s)$ agrees with the functions associated with lines in T_1 at all $s \in S$ (since we already argued that these functions are unique if they exist at all).

In general, each MV line $L_i \in T_i$ intersects the union of the MV lines in T_{i-1} at exactly $L_i(S)$. So by the same argument, for each $L_i \in T_i$, there is at most one function $g_{L_i} \in Z$ for which $g_{L_i}(s)$ agrees with the functions associated with lines in T_{i-1} at all $s \in S$.

We conclude that if S is "good," then there is at most one collection of functions that passes the MV consistency test, as required. $\square$

6 The reduction

Recall that the proof that a construction is indeed a HSG takes the form of a protocol for computing the hard function if the HSG fails. We will specify a commit-and-evaluate protocol $\pi = (\pi_{\text{commit}}, \pi_{\text{eval}})$ that takes advice $\alpha = D$ (where D is a co-nondeterministic circuit) and attempts to compute the polynomial p. We will prove that whenever D catches the generator $\text{RMV}_{h,d}(p)$ then the protocol π conforms with p resiliently. (Note that this does not mean that π *computes* p. However, in our application we will be able to use π to construct a protocol that *does compute* p). Our main theorem is stated below. In fact, following [GSTS03], we prove a slightly stronger statement in which the resiliency of the protocol follows regardless of whether D catches $\text{RMV}_{h,d}(p)$ as long as D rejects few inputs. This will be useful later on.

Theorem 6.1. *Let d, h, r, m, q be as in Figure 1. Let $p : \mathbb{F}_q^r \to \mathbb{F}_q$ be a polynomial of degree at most h. Then there is a commit-and-evaluate protocol $\pi = (\pi_{commit}, \pi_{eval})$ with advice $\alpha = D$, where D is a co-nondeterministic circuit of size poly(m), that satisfies:*

Conformity *If D rejects every element of $RMV_{h,d}(p)$ then π conforms with p.*

Resiliency *If D rejects at most a $1/3$-fraction of its inputs then π is resilient.*

Efficiency *π runs in time $h^{O(d \log_d r)}$ and has $\log_d r$ rounds.*

Moreover, π_{eval} has completeness 1, and π_{commit} is a two round protocol.

The rest of Section 6 is devoted to the proof of Theorem 6.1. Our main results (Theorems 2.4 and 2.5) then follow from Theorem 6.1 largely using machinery already worked out in [GSTS03].

6.1 The recursive commit-and-evaluate protocol

In Figure 3 we formally present the protocol σ used to prove Theorem 6.1, incorporating the ideas discussed in Section 5.1. Our main lemma regarding this protocol is:

Lemma 6.2 (correctness of σ). *Let d, h, r, m, q be as in Figure 1. Let $p : \mathbb{F}^r \to \mathbb{F}$ be a polynomial of degree at most h. Let D be a co-nondeterministic circuit of size poly(m). Let $\tau = (\tau_{commit}, \tau_{eval})$ be a commit-and-evaluate protocol such that for every MV line L, τ with advice (D, L) conforms resiliently to p_L (with completeness 1, soundness 2^{-10v^d} and resiliency $1 - 2^{-10v^d}$). Furthermore assume that τ_{commit} is a two round protocol. Then the following hold:*

Conformity *If D rejects every element of H_p then σ with advice D conforms with p with completeness 1.*

Resiliency *If D rejects at most $1/3$ of its inputs then σ with advice D is $9/10$-resilient, with the soundness error set to 1/10.*

Efficiency *If τ runs in time t and has $2i$ rounds then σ runs in time $th^{O(d)}$ and has $2(i+1)$ rounds. Furthermore, σ_{commit} is a two round protocol.*

Our protocol is paired with the construction in Figure 1 and uses the parameters of that construction.

Ingredients The protocol relies on commit-and-evaluate protocol τ, such that for every MV line L, τ with advice (D, L) conforms resiliently with the function p_L. In the final proof, this protocol will exist by recursion.

Operation of the commit phase σ_{commit}

- The input is $1^{\log |\mathbb{F}_q^r|}$ and the auxiliary input is the co-nondeterministic circuit D.
- Arthur sends a random set $S \subseteq B$ of size $v = k^2$.
- Merlin replies with an S-box $a : S^d \to \mathbb{F}$.
- Arthur outputs (S, a, D).

Operation of the evaluation phase σ_{eval}

- The input is $x \in B^d (= \mathbb{F}_q^r)$, and the auxiliary input is the output of the commit phase.
- Arthur and Merlin compute $T_1, \ldots, T_d$, the MV path to x using S. For each MV line L on the MV path (note that the number of such lines is bounded by v^d) we perform the following actions (in parallel for all lines L):

  Inner commitment Arthur and Merlin play the commit protocol τ_{commit} with advice (D, L) which outputs a commitment c_L.
  At this point Arthur and Merlin hold the auxiliary input c_L required to play the evaluation protocol τ_{eval} for the MV line L on any input $z \in B$. To simplify the notation we use τ_L below as if it were a function, with the understanding that any "function evaluation" $\tau_L(z)$ actually invokes the evaluation protocol τ_{eval} on input $z \in B$, with commitment c_L as its auxiliary input. Note that if the commit-and-evaluate protocol τ is resilient, then with high probability over the randomness of the commit phase, τ_{eval} is indeed a fixed function when given the commitment c_L.

  Low-degree test Let M_{LDT} be the machine associated with the low degree test of Lemma 3.2 with $\epsilon = 1/10$ and $\delta = 2^{-10v^d}$ (which can be achieved by amplification as explained in Section 3.2). Arthur chooses randomness for M_{LDT}, and then Arthur and Merlin run $M_{\text{LDT}}^{\tau_L}$ with that randomness. If the low degree test fails then Arthur stops and outputs $\perp$.
  If we get to this point in the protocol, we are ensured (with high probability) that τ_L is close to a low-degree polynomial. We would like to access that nearby low-degree polynomial for the remainder of the protocol, and we will use self-correction for that purpose. Let M_{SC} be the machine associated with the self-corrector of Lemma 3.4 using $\epsilon = 1/5$ and $\delta = 2^{-10v^d}$ (again this can be achieved by amplification).

  Small-set test Arthur chooses at random seeds $y_1, \ldots, y_{100v^d}$ for the local C-extractor E, and then Arthur and Merlin compute $w_{L,j} = E^{M_{\text{SC}}^{\tau_L}}(y_j)$. Finally Merlin supplies witnesses showing that for all j, D rejects $w_{L,j}$.

- **MV consistency test:** Arthur and Merlin perform the evaluations (using the self-corrector M_{SC}) of the various τ_L required for the MV consistency test (see Figure 2), with input x, S, the S-box a. (By that we mean that $M_{\text{SC}}^{\tau_L}$ plays the role of the function g_L needed for performing the test.)
- Arthur stops and outputs $\perp$ if any of the tests fail. Otherwise, Arthur and Merlin compute $w = M_{\text{SC}}^{\tau_L}(x_d)$ where L is the single MV line in T_d, and Arthur outputs w.

Figure 3: Commit-and-evaluate protocol σ with advice D, for use with recursive MV generator $\text{RMV}_{h,d}(p)$

Proof. (**Conformity**) In the commit phase σ_{commit} Arthur sends a set $S \subseteq B$ of size v and Merlin replies with an S-box a. We need to show that for every choice of set $S \subseteq B$ of size v there exists an S-box a and a Merlin strategy M for σ_{eval} such that $\text{out}(\sigma_{\text{eval}}, M, x, (S, a, D)) = p(x)$. Fix some subset S of size v. Define an S-box a by $a(s_1, \ldots, s_d) = p(s_1, \ldots, s_d)$. Merlin will send a in σ_{commit}. We define a collection of polynomials $g_L = p_L$, one for each line L in the MV path to x using S. By Lemma 5.4 the MV consistency test passes with these choices. We now define a Merlin strategy for σ_{eval} as follows: in the inner commitment step, Merlin will "play honestly" and use the strategy that guarantees that that τ_{eval} conforms with p_L given the commitment c_L generated in the commit phase. (Note that Merlin has such a strategy which succeeds with probability one by the conformity of τ). Merlin can now pass the low degree test by simply following the protocol (as p_L is indeed a low degree polynomial). For the small-set test, we notice that by assumption D rejects all elements of H_p and so D rejects $E^{p_L}(y)$ for every MV line L and every seed y. Thus, Merlin can pass the small-set test. Finally we observe using Lemma 5.4 that the output w when Merlin follows this strategy is indeed $p(x)$ as required. Note that the strategy we described succeeds with completeness 1.

(**Resiliency**) We need to show that for a uniformly chosen set $S \subseteq B$ of size v, with high probability, for every S-box a the protocol σ_{eval} is PSV when played with auxiliary input (S, a, D). The protocol σ_{eval} invokes the commitment protocol τ_{commit} once for every MV line on the MV path and there are at most v^d such MV lines. We now argue that by our requirement on the resiliency of τ we have that with probability greater than $99/100$ over the coin tosses of Arthur in the invocations of τ_{commit}, all commitments c_L obtained in the inner commitment step have the property that τ_{eval} with auxiliary input c_L is PSV. To see that we note that τ_{commit} is a two round protocol and therefore with probability $1 - 2^{-10v^d}$ over Arthur's choice of coins, every reply of Merlin results in a commitment string on which τ_{eval} is PSV. It follows by a union bound that with probability at least $99/100$ all coin tosses of Arthur in the invocations of τ_{commit} have the aforementioned property.[7] From now on we assume that this event happens and this allows us to think of τ_L (the invocation of τ_{eval} with auxiliary input c_L) as functions. Note that Merlin still has the liberty to play any strategy that he wants in the commitment phase of τ and thus has many different choices for what partial function to commit to. We will show that there is at most one choice that passes all tests.

We claim that if Arthur does not halt during the low-degree test step then with probability larger than $99/100$ (over Arthur's random choices for the low-degree test), every line L on the MV path is close to a polynomial g_L of degree at most h, and the self-corrector M_{SC} accesses this g_L. The follows from a union bound and the fact that the error for the low-degree test is at most 2^{-10v^d}.

We now define the set Z to be all polynomials $g : \mathbb{F}^{r/d} \to \mathbb{F}$ such that $\Pr_y[D \text{ rejects } E^g(y)] > 1/2$. We use Proposition 3.8 to argue that $|Z| \leq 2^k = 2^{h^5}$. This follows by having the set D of Proposition 3.8 be the set of inputs on which co-nondeterministic circuit D rejects, and by noticing that $|D|/2^m + \epsilon \leq 1/2$ (where here $\epsilon = 1/k < 1/10$ is the error of the extractor E).

We claim that if Arthur does not halt during the small-set test then with probability larger than $99/100$, for every L on the MV path, $g_L \in Z$. This is because if $g_L \notin Z$ then the probability (over the choice of seeds for the extractor and randomness for the self corrector) that the small set test passes on L is at most 2^{-5v^d} and by a union bound over all MV lines in the MV path we have that the probability that this event occurs for any L on the MV path $< 1/100$.

Finally, we claim that if Arthur does not halt during the MV consistency test then by Lemma 5.5 there is at most one choice for a collection of functions from Z that pass the MV consistency test. To use the Lemma we must check that $v = |S|$ is large enough compared to $K = 2^k$ which is the bound we have on

[7] We remark that although the argument above uses the fact that τ_{commit} is a two round protocol, the proof also goes through without this assumption.

the size of Z. Indeed, taking $\gamma = 99/100$ and $\beta = 1/10$ we have that

$$|S| = v \geq (2 \log K + \log(1/(1-\gamma)))/ \log(1/\beta)$$

as required. We conclude that any Merlin strategy on which Arthur does not halt and output $\perp$, with probability at least $9/10$ must end up defining this unique collection of functions g_L. In particular, there is at most one choice for the function g_L for the single MV line $L \in T_d$, and as this function defines the output uniquely, there is at most one possible value that Arthur can output and the protocol is resilient with probability 9/10 and soundness $1/10$.

(**Efficiency**) We go over the steps one by one. The MV path contains at most v^d MV lines. For each such MV line Arthur and Merlin perform computations that take time $\text{poly}(q, r, m, v^d) \leq h^{O(d)}$ when given oracle access to τ_L. Thus, overall the running time of σ is bounded by $tv^d h^{O(d)} = th^{O(d)}$. We now turn our attention to the number of rounds. The number of rounds of protocol σ_{eval} is precisely the number of rounds of τ. This is because to actually execute σ_{eval}, Arthur picks all the randomness for the various low-degree tests, and the randomness to run the self-corrector on the evaluations required for all the other tests. Then Arthur and Merlin play all the requested invocations of $\tau_L(z)$ for the various lines L and evaluation points z, in parallel. Merlin includes witnesses for the various small-set tests in his final message to Arthur.

Finally, we note that σ_{commit} has two rounds and therefore the total number of rounds of σ is the number of rounds of τ plus two as required. ☐

6.2 Proof of the main technical theorem

We now show that Theorem 6.1 follows from Lemma 6.2.

Proof. (of Theorem 6.1) Let D be a co-nondeterministic circuit. Recall that we only allow polynomials $p : \mathbb{F}^r \to \mathbb{F}$ with $r = d^j$ where $j \geq 1$ is an integer. We prove the Theorem by induction on j.

(**Base case**) We start with the base case $j = 1$. In this case the output of $\text{RMV}_{h,d}(p)$ is simply H_p. For the base case to follow from Lemma 6.2 we only need to supply a commit-and-evaluate protocol τ meeting the requirements in Figure 3. We use the trivial protocol in which Merlin sends to Arthur a polynomial (by specifying all coefficients) that is supposed to be p_L. More formally, in the commit protocol τ_{commit} Arthur sends nothing and Merlin replies with a string a that encodes a polynomial $g_L : \mathbb{F} \to \mathbb{F}$ (the honest Merlin will send p_L). In the evaluation protocol τ_{eval} Arthur can evaluate g_L on an input by himself without the help of Merlin. It is immediate that this protocol τ_{eval} meets the requirements of Figure 3 and the assumptions of Lemma 6.2, and therefore the base case follows. Note that τ is a two round protocol (actually it is a nondeterministic protocol rather than an AM protocol as Arthur does not send any random messages). Furthermore, note that τ runs in time $\text{poly}(h)$.

(**Induction step**) Let $j > 1$. We assume by induction that we already have a commit-and-evaluate protocol $\tau = (\tau_{\text{commit}}, \tau_{\text{eval}})$ that meets the requirements of Theorem 6.1 for every p over $r = d^{j-1}$ variables. Furthermore, we assume by induction that τ has completeness 1 and that τ_{commit} is a two round protocol. We observe that such a protocol meets the requirements of Figure 3 as well as the requirements of Lemma 6.2. This follows because we can amplify soundness and resiliency errors to the level required in Lemma 6.2 with slowdown $v^{O(d)} = h^{O(d)}$. Furthermore, for the conformity part we observe that since D rejects every element of $\text{RMV}_{h,d}(p)$ it in particular rejects every element in H_p. Thus, the induction step follows from Lemma 6.2. Any recursive level multiplies the running time by a factor of $h^{O(d)}$ and adds two rounds. There are $\log_d r$ such recursion levels and the Theorem follows. ☐

7 Obtaining our main results

In this section we show how our main results (Theorems 2.4, 2.5, and 2.6) follow from Theorem 6.1. The argument for this part is essentially the argument in [GSTS03] except that now we use the new generator RMV (Figure 1) rather than the generator of [MV05]. We give a high level overview of the argument in the next subsection. For completeness we also provide a full formal proof that appears in the remainder of this section.

7.1 High level overview of the argument

In this subsection we give a high level overview of how to obtain our main theorems. We start with Theorem 2.4. Let f be the characteristic function of a language complete for E that is instance-checkable (via Theorem 4.5). We are given a function $s = s(\ell)$. Fix ℓ and set $m = s(\ell)^{\Theta\left(1/(\log \ell - \log\log s(\ell))^2\right)}$. Consider a language L in AM and let σ be a (standard, two round) AM protocol for L with perfect completeness (without loss of generality [FGM$^+$89]). We will design a nondeterministic machine M running in time exponential in ℓ and show that if for each ℓ, M does not agree with L on an m-bit string x produced by a uniform nondeterministic procedure R ("the derandomization fails on feasibly generated inputs"), then f can be *computed* by an AM protocol running in time $s(\ell)$. We start by defining the machine M, which uses the generator RMV from Figure 1:

7.1.1 The generator and the derandomization

Set $h = m^{100}$, $q = h^{100}$, and set d to be a large constant and $r = O(\ell/\log h)$. It is standard that there is a polynomial $p : \mathbb{F}^r \to \mathbb{F}$ (the *low degree extension* [BF90]) of total degree at most h over $r = O(\ell/\log h)$ variables such that for every $y \in \{0,1\}^\ell$, $p(y)$ equals $f(y)$. Furthermore, the coefficients of p can be computed in time $2^{O(\ell)}$.

Given the polynomial p we run $\mathrm{RMV}_{h,d}(p)$ in time $q^{O(r)} = 2^{O(\ell)}$ to generate a set H of at most $q^{O(r)} = 2^{O(\ell)}$ strings of length m. The nondeterministic machine M is defined as follows: for every string $z \in H$ we simulate the protocol σ on x with z used as Arthur's randomness and guess an answer for Merlin. We accept if all of the simulated runs of the protocol accept. Note that M is indeed a nondeterministic machine that runs in time exponential in ℓ.

7.1.2 The reduction

Assume that M disagrees with L on x. Because σ has perfect completeness this can only happen when $x \notin L$ and yet M accepts x. Define the co-nondeterministic circuit $D_x(y)$ that rejects if on input x, Merlin has a reply to Arthur's message y (in the AM protocol σ for L) that causes Arthur to accept. By the efficiency of protocol σ, D_x has size $\mathrm{poly}(m)$, and by the soundness of protocol σ, we have that D_x rejects at most a third of its inputs. Finally since M accepted x, D_x must reject every $y \in H$.

Note that we can now use the protocol $\pi = (\pi_{\text{commit}}, \pi_{\text{eval}})$ of Theorem 6.1 with advice D_x and we get that π conforms with p resiliently and runs in time $m^{O(d \log_d r)}$. However, the protocol π only conforms resiliently with p and does not necessarily *compute* f, as discussed at the end of Section 4.1.

7.1.3 Using instance checkers

To solve this problem we will use instance checkers (in the same way they are used in [BFNW93, TV02, GSTS03]). Recall that we chose a function f that has an instance checker. For an instance-checkable f, a "commit and evaluate" protocol that conforms resiliently with f can be converted into a standard AM protocol for f:[8]

Theorem (informal) 7.1. *Let f be a function that is instance checkable. Let $\pi = (\pi_{commit}, \pi_{eval})$ be a commit-and-evaluate protocol that conforms with f resiliently. Then there is an AM protocol π' that* computes *f and runs in time comparable to that of π_{eval} using two additional rounds.*

Proof. (sketch) We describe the AM protocol π'. Given input x, Arthur and Merlin execute the commitment protocol π_{commit}. By the resiliency of π following this phase with high probability Merlin is committed to some (partial) function g (which may be different from f). Arthur chooses randomness for the instance checker and sends it to Merlin. The two parties then simulate the instance checker on input x where oracle calls are simulated by playing the evaluation protocol π_{eval}. Arthur outputs the recommendation of the instance checker regarding the value of $f(x)$. The theorem follows immediately from the properties of instance checkers. $\square$

We conclude that f has an AM protocol that computes it in time $m^{O(\log r)}$ that uses $O(\log r)$ rounds (recall that d is a constant). This protocol can be transformed into a two round protocol running in time $m^{O(\log^2 r)}$ and the parameters are set so that the time is at most $s(\ell)$ as required. Thus, we obtain a two round AM protocol that computes an E-complete problem in time $s(\ell)$.

7.1.4 The case of AM $\cap$ coAM

We now explain the idea for Theorem 2.5. A natural idea to remove the restriction to feasibly generated inputs is to have Merlin supply the input x (rather than having it supplied by some external uniform procedure R). The only part of the above argument that might fail is that we can no longer be sure that D_x rejects at most $1/3$ of its inputs, and then the resiliency of the protocol π is not guaranteed. However, if Arthur can verify that $x \notin L$, then the corresponding circuit D_x must reject at most a third of its inputs, and the resiliency of π follows. In general, Arthur has no way to check that $x \notin L$, but when $L \in \text{AM} \cap \text{coAM}$ Merlin can convince Arthur that $x \notin L$.

In the next three subsection we give the precise details for the argument outlined in Section 7.1.

7.2 Nondeterministic simulation of AM protocols

We start with describing how to use a hitting-set generator against co-nondeterministic circuits to perform nondeterministic simulation of AM protocols. This is standard, but we go through it in order to set parameters for the next part. The first observation is that given an AM language L and an input x, the behavior of the AM protocol on x can be captured by a co-nondeterministic circuit D_x which receives the random coin tosses of Arthur as input.

Lemma 7.2. *For any language $L \in AM$ there is a constant $c > 0$ such that for any input $x \in \{0,1\}^n$ there is a co-nondeterministic circuit D_x of size $m = n^c$ such that:*

- *If $x \in L$ then D_x rejects all inputs.*
- *If $x \notin L$ then D_x rejects at most a $1/3$-fraction of its inputs.*

[8] A technicality is that instance checkers may query inputs that are longer than their input. As a result some care is needed when stating the next theorem formally. The precise details appear in the formal proof.

- *Circuit D_x can be produced in polynomial time from x.*

Proof. By [FGM$^+$89] we can assume that the AM protocol for L has perfect completeness. Consider the following deterministic circuit $D_x(y, a)$: simulate Arthur's computation with coin toss y and Merlin's response a and flip the final answer. This deterministic circuit can be interpreted as a co-nondeterministic circuit $D_x(y)$ that fulfils all the requirements above.[9] ☐

When given a hard problem we use the low-degree extension [BF90] to transform it into a low degree polynomial as follows:

Lemma 7.3 (low degree extension). *Let $f : \{0,1\}^* \to \{0,1\}$ be a function, ℓ be an integer, $h \leq \ell^{O(1)}$ be a prime power, and $q = h^{O(1)}$. There is a polynomial $p : \mathbb{F}^r \to \mathbb{F}$ of total degree h over a field $\mathbb{F}$ of size q with $r = O(\ell/\log h)$ variables such that:*

- *There is an injective mapping $I : \{0,1\}^\ell \to \mathbb{F}^r$ that is computable in polynomial time.*
- *For every $y \in \{0,1\}^\ell$, $f(y) = p(I(y))$.*
- *p can be computed in time $2^{O(\ell)}$ when given oracle access to f.*

The polynomial p is called the low-degree extension of f at length ℓ with degree h.

The proof of Lemma 7.3 is standard (see e.g., [Uma03]).

To prove Theorems 2.4, 2.5, and 2.6 we need to construct a nondeterministic machine that attempts to simulate a given AM language L. Figure 4 describes how to use the generator RMV from Figure 1 to construct such a nondeterministic machine M_L. We observe the following properties of the machine M_L.

Lemma 7.4. *Let L be a language in AM, and let M_L, f, ℓ, and $v(\ell)$ be as specified in Figure 4.*

- *If f is computable in time $2^{\ell^{O(1)}}$ and $v(\ell) = \ell^{O(1)}$ then the machine M_L runs in nondeterministic time $2^{\ell^{O(1)}}$ on inputs of length m.*
- *If f is computable in time $2^{O(\ell)}$ and $v(\ell) = O(\ell)$ then the machine M_L runs in nondeterministic time $2^{O(\ell)}$ on inputs of length m.*
- *If x is an input on which $L(M_L)$ and L disagree, then $x \notin L$.*

Proof. We have that $m < s(\ell) \leq 2^\ell$. Therefore we can neglect operations that take time polynomial in m. The machine M_L needs to compute the low degree extension p of f. By Lemma 7.3 this takes time $2^{O(\ell)}$ when given oracle access to f. The other main factor in the running time is computing RMV. By Lemma 5.2 this takes time $q^{O(r)} = 2^{O(v(\ell))}$, given p. The first two items of the lemma follow. The third item follows from Lemma 7.2 as for every $x \in L$ we have that D_x rejects all inputs and in particular rejects all outputs of RMV. ☐

To finish up the argument and prove our main theorems we show that given an input x on which M_L fails to simulate L correctly we can give an AM protocol for the supposedly hard function f. This is done in the next subsection.

[9] It is indeed more natural to think of D_x as a nondeterministic circuit (without flipping the answer). The reason we speak about co-nondeterministic circuit is that the definition of hitting set generators is not symmetric in zeroes and ones and in order to meet this definition we need to flip the output. In this choice we follow [MV05, GSTS03].

Our procedure uses the construction and parameters of Figure 1.

Ingredients

- An AM language L. This is the language to be derandomized.
- A function $f : \{0,1\}^* \to \{0,1\}$. This is the "hard function" supplied to the derandomization procedure.
- An time constructible integer function $\ell \leq s(\ell) \leq 2^\ell$ (see definition 2.3). This is a function which measures how hard is the function f.
- An integer function $v(\ell) \geq \ell$. The function $v(\ell)$ determines the length of queries made by an instance checker for f on inputs of length ℓ. By Theorem 4.5 we have that $v(\ell) = \ell^{O(1)}$ when f is computable in EXP and $v(\ell) = O(\ell)$ when f is computable in E.

Parameter: A constant c'.

Input: A string x of length n.

Operation of M_L on input x

- Let c be the constant guaranteed by Lemma 7.2 for L and set $m = n^c$.
- Compute the smallest integer ℓ such that $s(\ell)^{c'/(\log v(\ell) - \log\log s(\ell))^2} \geq m$. Since s is time-constructible, this can be found efficiently by binary search. Note that this is exactly the relationship between s, ℓ and m and n that we need to fulfill in our main theorems. We can assume without loss of generality that $m^{100} \geq v(\ell)$. Otherwise M_L can just decide L by brute-force simulation in time $2^{O(n^c)} = 2^{O(m)}$ which is at most $2^{O(\ell)}$ if $m^{100} \leq v(\ell)$.
- Set h to be the smallest prime power larger than m^{100} and $q = h^{100}$. Let p be the low degree extension of f at length $v(\ell)$ over the field with q elements. We have that $p : \mathbb{F}^r \to \mathbb{F}$ is a polynomial with $r = O(v(\ell)/\log h)$ variables over a field of size q.
- Set $d = 2$ and compute $H = \mathrm{RMV}_{h,d}(p)$, which is a multi-set of m bit strings.
- For every string $z \in H$ guess a witness showing that $D_x(z)$ rejects. Recall that D_x is a co-nondeterministic circuit, so it has short witnesses for rejection.
- Finally, accept x if and only if for every $z \in H$ the guessed witness proves that $D_x(z)$ rejects.

Figure 4: The nondeterministic machine $M_L(x)$ which attempts to decide the AM language L.

7.3 Establishing the correctness of the nondeterministic simulation

We now suppose that the machine M_L disagrees with the AM language L on some input x and show how the protocol π from Theorem 6.1 yields an AM protocol that computes the function f on all inputs of a particular length that is a function of the length of x. We will use the fact that problems complete for E or EXP have instance checkers. In Figure 5 we present the AM protocol τ for computing the function f in the event that M_L fails to decide L.

Our main theorem of this subsection asserts that protocol τ indeed computes f on all inputs of a particular length when supplied with an advice string x on which M_L disagrees with the language L. In fact, we

Our protocol refers to the parameters and ingredients of the procedure in Figure 4.

Ingredients

- An instance checker IC for f that makes queries of length $v(\ell)$ on inputs of length ℓ.
- The commit and evaluate protocol $\pi = (\pi_{\text{commit}}, \pi_{\text{eval}})$ that is guaranteed in Theorem 6.1 when using the polynomial p as defined in the construction of the machine M_L using the parameters d, h, r, m, q defined there. Recall that p is the low degree extension of f at length $v(\ell)$ and that protocol π expects as advice a co-nondeterministic circuit of size poly(m).

Input: A string y of length ℓ. The protocol is trying to compute $f(y)$.

Auxiliary input: A string x of length n.

Operation

- Arthur computes the circuit D_x defined in Lemma 7.2.
- Arthur and Merlin play the commit phase π_{commit} using advice string D_x and they obtain as output a commitment com.

  At this point Arthur and Merlin hold the auxiliary input com required to play the evaluation protocol π_{eval} on any input to p. Note that as p is the low-degree extension of f at length $v(\ell)$ we can use π_{eval} to evaluate f at any input of length $v(\ell)$ by using the mapping I from Lemma 7.3. Furthermore, we will show that protocol π conforms resiliently with p and therefore the reader can imagine that π_{eval} is a fixed function when used with the auxiliary input com.
- Arthur chooses random coin tosses for the instance checker IC when run on input y and sends them to Merlin.
- Merlin simulates the run of IC on y using oracle f. Merlin sends the transcript of this simulation to Arthur and for all queries y' of length $v(\ell)$ made to f, Arthur and Merlin play the protocol π_{eval} on the input y' (in parallel). Arthur verifies that the output he obtains is consistent with the answer to the query provided by Merlin on y' in the transcript that Merlin sent. Arthur also verifies that the transcript is indeed valid when using the supplied oracle queries and answers. Arthur outputs $\perp$ and halts if he detects any inconsistency.
- Arthur outputs the output of $IC(y)$ that appears in the transcript.

Figure 5: The protocol $\tau(y)$ which attempts to compute $f(y)$

will prove a stronger statement in which the soundness of τ holds under the weaker condition that $x \notin L$. (This is indeed a weaker condition by Lemma 7.4). This stronger statement will be helpful later on when proving Theorem 2.5 and 2.6.

Theorem 7.5. *Protocol τ in Figure 5 satisfies:*

completeness *If the machine M_L does not agree with L on input x then τ with auxiliary input x conforms with f on inputs of length ℓ with completeness* 1.

soundness *If $x \notin L$ then τ with auxiliary input x is PSV on inputs of length ℓ.*

efficiency *Protocol τ runs in time $m^{O(\log v(\ell) - \log\log m)}$ and has $O(\log v(\ell) - \log\log m)$ rounds.*

Proof. The three items follow directly from Theorem 6.1; the details appear below:

(Completeness) We have that L and $L(M_L)$ disagree on x. By Lemma 7.4 it follows that $x \notin L$. We conclude that M_L did accept x and in particular we have that D_x rejects all the elements z in $H = \mathrm{RMV}_{h,d}(p)$. By Theorem 6.1 we have that the protocol π with advice D_x conforms with p, with completeness 1. Therefore, Merlin has a strategy for π_{commit} so that the commitment string com obtained by Arthur is such that π_{eval} with auxiliary input com conforms with p with completeness 1. Thus, by simulating the instance checker IC correctly, Merlin can lead Arthur to output $f(y)$ as he can convince Arthur that the transcript of the instance checker is correct.

(Soundness) By Lemma 7.2 if $x \notin L$ then D_x rejects at most a $1/3$-fraction of its inputs. By Theorem 6.1 we have that in this case π is resilient. It follows that no matter how Merlin plays in the commit phase π_{commit} the output com is such that π_{eval} with auxiliary input com is PSV. It follows that there exists a function $g : \{0,1\}^{v(\ell)} \to \{0,1\}$ such such that for any input y' to π_{eval} no matter how Merlin plays he cannot lead Arthur to accept a value different than $g(y')$ with noticeable probability. In such a case (assuming π is sufficiently amplified using Proposition 4.3) we have that no instantiation of π_{eval} in the protocol τ answers incorrectly. By the properties of instance checkers we have that when $IC(y)$ is run with oracle access to g then with high probability (over the randomness for IC supplied by Arthur) the output is either $f(y)$ or $\perp$. It follows that if Merlin is able to complete the execution of τ then with high probability Arthur outputs $f(y)$. Thus, the probability that Merlin can make Arthur output a value different than $f(y)$ is smaller than the default soundness error of $1/3$.

(Efficiency) Computing D_x can be done in time $\mathrm{poly}(n) = m^{O(1)}$ by Lemma 7.2. By Theorem 6.1 the protocol π runs in time $h^{O(d \log_d r)}$. Recall that $d = O(1)$, $h = m^{\Theta(1)}$ and $r = O(v(\ell)/\log h)$. It follows that the running time is bounded by $m^{O(\log v(\ell) - \log\log m)}$. The instance checker runs in time $\ell^{O(1)} \leq m^{O(1)}$ and therefore the number of queries (which controls the number of invocations of π_{commit} is bounded by $m^{O(1)}$. Overall, the running time of τ is indeed $m^{O(\log v(\ell) - \log\log m)}$ as required. All the invocations of π_{eval} are done in parallel and therefore τ has only two additional rounds over π_{eval} and the total number of rounds is $O(\log_d r) = O(\log v(\ell) - \log\log m)$. ☐

7.4 Putting everything together

We are finally ready to prove Theorems 2.4, 2.5, and 2.6. The setup and parameter choice for the three theorems is very similar so we will start by describing the common part of the three proofs.

The setup and parameters: Let $s(\ell)$ be an integer function satisfying the requirements of Theorems 2.4, 2.5, and 2.6. Let L be a language in AM. Let f be a characteristic function of a problem in E (resp. EXP) that has an instance checker IC that makes queries of length $v(\ell) = O(\ell)$ (resp. $v(\ell) = \ell^{O(1)}$)). Note that the existence of such a function f is guaranteed by Theorem 4.5. Let M_L be the nondeterministic machine defined in Figure 4. We first verify that the relationship between the parameters n, ℓ and $s(\ell)$ are exactly as specified in the theorems.

Recall that M_L receives inputs of length n and the description of M_L fixes the parameter $m = n^c$ (where c is a constant that depends only on the AM language L). Also recall that M_L chooses ℓ as a function of

m. More precisely, we choose ℓ to be the smallest integer such $s(\ell)^{c'/(\log v(\ell)-\log\log s(\ell))^2} \geq m$ where the constant $c' > 0$ is a parameter. Thus, we have that $n = s(\ell)^{\Theta(1/(\log v(\ell)-\log\log s(\ell))^2)}$ as required (where the constants hidden inside the Θ depend only on the constants c, c'). Note that by Lemma 7.4 the machine M_L runs in nondeterministic time $2^{O(\ell)}$ (resp. $2^{\ell^{O(1)}}$) on inputs of length n. Thus, our choice of parameters is as promised in Theorems 2.4, 2.5, and 2.6.

In the proofs of the three Theorems we need to show that if M_L fails to decide L (where the meaning of this statement differs in the different theorems) then there is an AM protocol that computes f and runs in time $s(\ell)$. Let τ be the Arthur-Merlin protocol defined in Figure 5. The high level idea is that by Theorem 7.5 we are guaranteed that τ indeed computes f when it is given an auxiliary input x on which L and $L(M_L)$ disagree. The difference between the three proofs is in how this string x is obtained. Before going into this issue let us first observe that the running time of τ is indeed smaller than $s(\ell)$ for our choice of parameters.

By Theorem 7.5 protocol τ runs in time $m^{O(\log v(\ell)-\log\log m)}$ and has $O(\log v(\ell) - \log\log m)$ rounds. Given an Arthur-Merlin protocol that runs in time T and has R rounds it is possible to collapse it into a two round protocol that runs in time $T^{O(R)}$ [BM88]. Thus we can get a two round protocol with running time to $m^{O(\log v(\ell)-\log\log m)^2}$. Recall that in the definition of M_L we chose ℓ to be the smallest integer such $s(\ell)^{c'/(\log v(\ell)-\log\log s(\ell))^2} \geq m$. Therefore we have:

$$m^{O(\log v(\ell)-\log\log m)^2} \leq \left(s(\ell)^{\frac{c'}{(\log v(\ell)-\log\log s(\ell))^2}} \right)^{O(\log v(\ell)-\log\log m)^2} \leq s(\ell)^{O(1)} \qquad (1)$$

where we are using the fact that $(\log v(\ell) - \log\log m) = O(\log v(\ell) - \log\log s(\ell))$ which follows because by the definition of m:

$$\begin{aligned}(\log v(\ell) - \log\log m) &= (\log v(\ell) - \log\log s(\ell)) + 2\log(\log v(\ell) - \log\log s(\ell)) + O(1) \\ &\leq (1 + o(1))(\log v(\ell) - \log\log s(\ell)) + O(1).\end{aligned}$$

We observe that the $O(1)$ in the exponent on $s(\ell)$ in Equation 1 (which depends on c', c and the hidden constants in Lemma 7.3 and Theorem 4.5) can be made to be any positive constant by choosing c' to be a sufficiently small constant.

We now split the proof into the cases of the three different theorems. We begin with the proof of Theorem 2.4. In this case there is an external machine R (the refuter) that supplies the auxiliary input x.

Proof. (of Theorem 2.4) Assume that M_L fails on feasibly generated inputs, and let R be a nondeterministic machine as in Definition 2.2. We are guaranteed that for all but finitely many input lengths n and for every accepting computation path, $R(1^n)$ outputs a string x of length n such that L and $L(M_L)$ disagree on x. We will show that f has a two round Arthur-Merlin protocol running in time $s(\ell)$ that computes f on inputs of length ℓ. This will prove Theorem 2.4.

Consider the following Arthur-Merlin protocol: When given input $y \in \{0,1\}^\ell$, Arthur and Merlin compute an integer n so that ℓ is the integer chosen by the nondeterministic machine M_L when given inputs x of length n. Merlin then sends a string x of length n with an accepting computation path of $R(1^n)$ that outputs x. The two parties then run the protocol τ on input y and auxiliary input x.

By the properties of the refuter R we have that L and $L(M_L)$ disagree on x. By Lemma 7.4 we have that $x \notin L$. By Theorem 7.5 this gives us the completeness and soundness properties of protocol τ with auxiliary input x. We conclude that the protocol above computes f on all but finitely many input lengths ℓ.

The running time of the Arthur-Merlin protocol above is dominated by the running time of τ which is bounded by $s(\ell)^{1/100}$. Thus, the entire protocol runs in time smaller than $s(\ell)$ as required. $\square$

In the case of Theorems 2.4 and 2.5 we have additionally that L is in coAM. When given $y \in \{0,1\}^\ell$ we will now rely entirely on Merlin to send a string x of length n that will be used as auxiliary input for the protocol τ. Unlike the case of Theorem 2.4, we do not have the refuter to ensure that Merlin indeed sends an $x \notin L$. We will therefore ask Merlin to also prove to Arthur that $x \notin L$ – which Merlin can do in this case because L is in coAM.

Proof. (of Theorem 2.5) We assume that L is also in coAM. We will show that if L and $L(M_L)$ disagree for all but finitely many input lengths n then f has a two round Arthur-Merlin protocol running in time $s(\ell)$ that computes f. This will prove Theorem 2.5.

Consider the following Arthur Merlin protocol: When given input $y \in \{0,1\}^\ell$, Arthur and Merlin compute an integer n so that ℓ is the integer chosen by the nondeterministic machine M when given inputs x of length n. Merlin sends a string x of length n (that is supposed to be a string on which L and $L(M_L)$ disagree). Arthur and Merlin then play the AM protocol for the complement of L on the input x (note that such a protocol exists as we are assuming that $L \in co$AM and we can assume without loss of generality that it has perfect completeness). By the completeness and soundness of this protocol at the end of this protocol Arthur is convinced with high probability that $x \notin L$. At this point Arthur and Merlin play protocol τ on input y using auxiliary input x.

An honest Merlin can indeed follow the protocol described above and using Theorem 7.5 it follows that Arthur will output $f(y)$ with probability one in this case. Furthermore, no matter how Merlin plays, Arthur will reject (with high probability) unless Merlin sends $x \notin L$, in which case the soundness of the protocol follows by Theorem 7.5.

The running time of this protocol is dominated by the running time of τ. Thus, the protocol can be collapsed into a two round protocol that runs in time $s(\ell)$ as required. □

For Theorem 2.6 we are interested in the case that M_L only fails on infinitely many input lengths n. In this case we would like the protocol for f to succeed on infinitely many input lengths ℓ. However, there is a subtle point here. In both protocols above we instructed Arthur and Merlin to compute n as a function of ℓ. Note however, that there are many lengths n which satisfy the relationship "n is an integer so that ℓ is the integer chosen by the nondeterministic machine M_L when given inputs x of length n". We were not concerned with this previously because all lengths n were good for our purposes. However, now only infinitely many lengths n are good. For this approach to work we need that for any length ℓ such that there is a good length n that satisfies the relation above, we can actually come up with such a length n.

We do not know how to do this in the setup of Theorem 2.4 (i.e., when the refuter only succeeds on infinitely many input lengths). However, we can do it in the setup of Theorem 2.5. We will now rely on Merlin to send such a length n. The soundness of the protocol for f still follows using Theorem 7.5 as Merlin still has to send an x that is not in L. However, the completeness is no longer guaranteed on all lengths ℓ as it is not necessarily the case that Merlin can come up with an n and an x such that L and $L(M_L)$ disagree on x. The formal proof appears below:

Proof. (of Theorem 2.6) We assume that L is also in coAM. We will show that if L and $L(M_L)$ disagree on infinitely many input lengths n then f has a two round Arthur-Merlin protocol running in time $s(\ell)$ such that on infinitely many input lengths the protocol computes f. This will prove Theorem 2.5. Note that there is no guarantee that there is a gap between completeness and soundness on "incorrect" lengths ℓ.

Consider the following Arthur Merlin protocol: When given input $y \in \{0,1\}^\ell$, Merlin sends an integer n and Arthur checks that ℓ is the integer chosen by the nondeterministic machine M_L when given inputs x of length n. Merlin then sends a string x of length n (that is supposed to be a string on which L and $L(M_L)$

disagree). From here on the proof is similar to that of Theorem 2.5; namely: Arthur and Merlin play the AM protocol for the complement of L on the input x. By the completeness and soundness of this protocol at the end of the protocol Arthur is convinced with high probability that $x \notin L$. At this point Arthur and Merlin play protocol τ on input y using advice x.

An honest Merlin can indeed follow the protocol described above (on infinitely many input lengths ℓ) and using Theorem 7.5 it follows that Arthur will output $f(y)$ with probability one in this case. Furthermore, no matter how Merlin plays Arthur will reject (with high probability) unless Merlin sends $x \notin L$ and the soundness of the protocol follows by Theorem 7.5. In fact, soundness is guaranteed on all lengths ℓ.

Again, the running time of this protocol is dominated by the running time of τ. Thus, the protocol can be collapsed into a two round protocol that runs in time $s(\ell)$ as required. □

8 Conclusions and open problems

In this paper we give improved uniform hardness versus randomness tradeoffs for Arthur-Merlin games that come very close to the "absolute low-end". A very natural open problem is to give a tradeoff that achieves the absolute low-end, namely, one that achieves $n = s(\ell)^{\Omega(1)}$ in Theorems 2.4, 2.5, and 2.6 rather than the current bound which gives $n = s(\ell)^{\Theta\left(1/(\log \ell - \log\log s(\ell))^2\right)}$ for E and $n = s(\ell)^{\Theta\left((1/\log \ell)^2\right)}$ for EXP. Our current results are suboptimal because of the following losses accumulated in the recursion:

- In the recursive AM protocol that is constructed in the proof of Theorem 6.1 every instantiation of the protocol at one level triggers poly(m) instantiations at the next level. As there are $\Theta\left(\log \ell - \log\log s(\ell)\right)$ levels we get that the running time of the protocol is $m^{\Theta(\log \ell - \log\log s(\ell))}$ rather than poly(m).
- Each recursive call also adds an additional round to the Arthur-Merlin protocol. At the end we also need to pay a penalty in the running time when collapsing the rounds to give a standard two round Arthur-Merlin protocol.

Another important open problem is to improve Theorem 2.4 so that the result holds for all inputs, rather than only inputs that are feasibly generated. Following [GSTS03] we already achieve such a clean statement for AM $\cap$ *co*AM. We remark that this can also be done for MA. As explained in [GSTS03], achieving this goal for AM, for the absolute low-end, will give an unconditional (although weak) derandomization of AM, placing it in a subexponential version of Σ_2^{P}.

9 Acknowledgements

We are grateful to Amnon Ta-Shma for helpful conversations, and the anonymous referees for their useful comments.

References

[ACR96] A. E. Andreev, A. E. F. Clementi, and J. D. P. Rolim. Hitting sets derandomize BPP. In *Automata, Languages and Programming, 23rd International Colloquium*, pages 357–368, 1996.

[ACRT99] A. E. Andreev, A. E. F. Clementi, J. D. P. Rolim, and L. Trevisan. Weak random sources, hitting sets, and BPP simulations. *SIAM Journal on Computing*, 28(6):2103–2116, 1999.

[AK01] V. Arvind and J. Köbler. On pseudorandomness and resource-bounded measure. *Theor. Comput. Sci*, 255(1-2):205–221, 2001.

[BF90] D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In *7th Annual Symposium on Theoretical Aspects of Computer Science*, volume 415 of *LNCS*, pages 37–48. Springer, 1990.

[BFL91] L. Babai, L. Fortnow, and C. Lund. Nondeterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1(1):3–40, 1991.

[BFLS91] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 21–31, 1991.

[BFNW93] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3(4):307–318, 1993.

[BK95] M. Blum and S. Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1):269–291, 1995.

[BM84] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, November 1984.

[BM88] L. Babai and S. Moran. Arthur-merlin games: A randomized proof system, and a hierarchy of complexity classes. *J. Comput. Syst. Sci.*, 36(2):254–276, 1988.

[FGM+89] M. Furer, O. Goldriech, Y. Mansour, M. Sipser, and S. Zachos. On completeness and soundness in interactive proof systems. *S. Micali, editor, Advances in Computing Research 5: Randomness and Computation*, pages 429–442, 1989.

[FS95] K. Friedl and M. Sudan. Some improvements to total degree tests. In *ISTCS*, pages 190–198, 1995.

[Gol98] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Springer-Verlag, Algorithms and Combinatorics, 1998.

[GS86] 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*, pages 59–68, 1986.

[GSTS03] D. Gutfreund, R. Shaltiel, and A. Ta-Shma. Uniform hardness versus randomness tradeoffs for Arthur-Merlin games. *Computational Complexity*, 12(3-4):85–130, 2003.

[IKW02] R. Impagliazzo, V. Kabanets, and A. Wigderson. In search of an easy witness: exponential time vs. probabilistic polynomial time. *J. Comput. Syst. Sci*, 65(4):672–694, 2002.

[Imp95] R. Impagliazzo. Hard-core distributions for somewhat hard problems. In *36th Annual Symposium on Foundations of Computer Science*, pages 538–545, 1995.

[ISW06] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Reducing the seed length in the Nisan-Wigderson generator. *Combinatorica*, 26(6):647–681, 2006.

[IW97] R. Impagliazzo and A. Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 220–229, 1997.

[IW98] R. Impagliazzo and A. Wigderson. Randomness vs. time: De-randomization under a uniform assumption. In *39th Annual Symposium on Foundations of Computer Science*, pages 734–743, 1998.

[Kab01] V. Kabanets. Easiness assumptions and hardness tests: Trading time for zero error. *J. Comput. Syst. Sci*, 63(2):236–252, 2001.

[KI04] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004.

[KvM02] A. R. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002.

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

[Lip90] R. J. Lipton. Efficient checking of computations. In *7th Annual Symposium on Theoretical Aspects of Computer Science*, pages 207–215, 1990.

[Lu04] C.-J. Lu. Encryption against storage-bounded adversaries from on-line strong extractors. *J. Cryptology*, 17(1):27–42, 2004.

[MV05] P. Bro Miltersen and N. V. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. *Computational Complexity*, 14(3):256–279, 2005.

[NW94] N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49(2):149–167, October 1994.

[Sha92] A. Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992.

[STV01] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62:236–266, 2001.

[SU05a] R. Shaltiel and C. Umans. Pseudorandomness for approximate counting and sampling. In *IEEE Conference on Computational Complexity*, pages 212–226, 2005.

[SU05b] R. Shaltiel and C. Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *Journal of the ACM*, 52(2):172–216, 2005.

[TV02] L. Trevisan and S. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. In *Proceedings of the 17th Annual Conference on Computational Complexity*, pages 129–138, 2002.

[Uma03] C. Umans. Pseudo-random generators for all hardnesses. *Journal of Computer and System Sciences*, 67:419–440, 2003.

[Uma05] C. Umans. Reconstructive dispersers and hitting set generators. In *APPROX-RANDOM*, pages 460–471, 2005.

[Vad04] S. P. Vadhan. Constructing locally computable extractors and cryptosystems in the bounded-storage model. *J. Cryptology*, 17(1):43–77, 2004.

[Yao82] A. C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23th Annual Symposium on Foundations of Computer Science*, pages 80–91, 1982.