Instance-Wise Hardness and Refutation versus Derandomization for Arthur-Merlin Protocols

Dieter van Melkebeek   Nicollas Mocelin Sdroievski

February 12, 2024

Abstract

A fundamental question in computational complexity asks whether probabilistic polynomial-time algorithms can be simulated deterministically with a small overhead in time (the BPP vs. P problem). A corresponding question in the realm of interactive proofs asks whether Arthur-Merlin protocols can be simulated nondeterministically with a small overhead in time (the AM vs. NP problem). Both questions are intricately tied to lower bounds. Prominently, in both settings *blackbox* derandomization, i.e., derandomization through pseudo-random generators, has been shown equivalent to lower bounds for decision problems against circuits.

Recently, Chen and Tell (FOCS'21) established near-equivalences in the BPP setting between *whitebox* derandomization and lower bounds for multi-bit functions against algorithms on almost-all inputs. The key ingredient is a technique to translate hardness into targeted hitting sets in an instance-wise fashion based on a layered arithmetization of the evaluation of a uniform circuit computing the hard function f on the given instance. Follow-up works managed to obtain full equivalences in the BPP setting by exploiting a *compression* property of classical pseudo-random generator constructions. In particular, Chen, Tell and Williams (FOCS'23) showed that derandomization of BPP is equivalent to *constructive* lower bounds against algorithms that go through a compression phase.

In this paper we develop a corresponding technique for Arthur-Merlin protocols and establish similar near-equivalences in the AM setting. As an example of our results in the hardness to derandomization direction, consider a length-preserving function f computable by a nondeterministic algorithm that runs in time n^a. We show that if every Arthur-Merlin protocol that runs in time n^c for $c = O(\log^2 a)$ can only compute f correctly on finitely many inputs, then AM is in NP. We also obtain equivalences between constructive lower bounds against Arthur-Merlin protocols that go through a compression phase and derandomization of AM via *targeted* generators. Our main technical contribution is the construction of suitable targeted hitting-set generators based on probabilistically checkable proofs of proximity for nondeterministic computations.

As a byproduct of our constructions, we obtain the first result indicating that whitebox derandomization of AM may be equivalent to the existence of targeted hitting-set generators for AM, an issue raised by Goldreich (LNCS, 2011). Byproducts in the average-case setting include the first uniform hardness vs. randomness tradeoffs for AM, as well as an unconditional mild derandomization result for AM.

1 Introduction

The power of randomness constitutes a central theme in the theory of computing. In some computational settings, randomness is indispensable for any algorithmic solution. In others, it is provably needed for attaining efficiency. In yet others, the use of randomness leads to algorithms that run faster than all known deterministic ones, but the question remains open: Does an efficient deterministic algorithm exist?

In the context of decision problems, the key question is whether probabilistic polynomial-time algorithms with bounded error (the class BPP) can be simulated deterministically with a small overhead in time. In the realm of interactive verification protocols, the corresponding question asks whether Arthur-Merlin protocols (the class AM) can be simulated nondeterministically with a small overhead in time. In both settings, polynomial overhead is conjectured to suffice but even subexponential overhead remains open. Both settings have intricate connections to the quest for lower bounds, referred to as hardness vs. randomness tradeoffs. In some cases equivalences are known. We first describe the situation for BPP and then the one for AM, the focal point of this paper.

BPP setting. The first hardness vs. randomness tradeoffs were developed for *blackbox* derandomization, where a pseudo-random generator (PRG) produces, in an input-oblivious way, a small set of strings that "look random" to the process under consideration on every input of a given length. A long line of research established tight equivalences between *blackbox* derandomization of prBPP (the promise version of the class BPP) and *nonuniform* lower bounds for exponential-time classes. At the low end of the derandomization spectrum, subexponential-time blackbox derandomizations of prBPP are equivalent to super-polynomial circuit lower bounds for EXP $\doteq$ DTIME$[2^{\text{poly}(n)}]$ [BFNW93]. At the high end, polynomial-time blackbox derandomizations of prBPP are equivalent to linear-exponential circuit lower bounds for E $\doteq$ DTIME$[2^{O(n)}]$ [IW97]. A smooth interpolation between the two extremes exists and yields tight equivalences over the entire derandomization spectrum [Uma03]. The results are also robust in the sense that if the circuit lower bound holds at infinitely many input lengths (equivalent to the separation EXP $\not\subseteq$ P/poly at the low end), then the derandomization works at infinitely many input lengths, and if the circuit lower bound holds at almost-all input lengths, then the derandomization works at almost-all input lengths.

A uniformization of the underlying arguments led to equivalences between derandomizations that work on *most* inputs of a given length, and *uniform* lower bounds, i.e., lower bounds against algorithms. This derandomization setting is often referred to as the *average-case* setting, where the underlying distribution may be the uniform one or any other polynomial-time sampleable distribution. At the low end, there exist subexponential-time simulations of BPP that work on all but a negligible fraction of the inputs of infinitely many lengths if and only if EXP $\not\subseteq$ BPP [IW01]. Unfortunately, the known construction does not scale well (see [TV07, CRTY20, CRT22] for progress toward an equivalence at the high end) and is not robust (a version for almost-all input lengths remains open). On the other hand, the result holds for blackbox derandomization as well as for general, "whitebox" derandomization, and implies an equivalence between blackbox and whitebox derandomization in this setting: If derandomization is possible at all, it can be done through pseudo-random generators.

This left open the setting of *whitebox* derandomizations that work for *almost all* inputs. For prBPP, such derandomizations are equivalent to the construction of *targeted* pseudo-random generators, which take an input x for the underlying randomized process, and produce a small set of strings that "look random" on that specific input x [Gol11]. Recently, Chen and Tell [CT21] raised the question of an equivalent lower bound condition, and proposed a candidate: *uniform* lower bounds for *multi-bit* functions (rather than usual decision problems) that hold on *almost-all inputs* in the following sense.

Definition 1 (Hardness on almost-all inputs). *A computational problem f is hard on almost-all inputs against a class of algorithms if for every algorithm A in the class there is at most a finite number of inputs on which A computes f correctly.*

Chen and Tell started from the following observation about derandomization to hardness at the high end of the spectrum.

Proposition 2 (Chen and Tell [CT21]). *If* $\text{prBPP} \subseteq \text{P}$*, then for every constant c there exists a length-preserving function f that is computable in deterministic polynomial time and is hard on almost-all inputs against* $\text{prBPTIME}[n^c]$.

Remarkably, they also established a converse, albeit with an additional uniform-circuit depth restriction on the hard function f. Their approach naturally yields a targeted *hitting-set generator* (HSG), the counterpart of a pseudo-random generator for randomized decision processes with one-sided error (the class RP and its promise version prRP).

Theorem 3 (Chen and Tell [CT21]). *Let f be a length-preserving function computable by logspace-uniform circuits of polynomial size and depth n^b for some constant b. If f is hard on almost-all inputs against* $\text{prBPTIME}[n^{b+O(1)}]$*, where $O(1)$ denotes some universal constant, then* $\text{prRP} \subseteq \text{P}$.

Note that the hardness hypothesis of Theorem 3 necessitates the depth n^b of the uniform circuits computing the function f to be significantly less than their size. Otherwise, there exists even a deterministic algorithm that computes f in time $n^{b+O(1)}$.

The proof of Theorem 3 constructs a polynomial-time targeted hitting-set generator for prRP, which generically implies a polynomial-time targeted pseudo-random generator for prBPP, and thus that $\text{prBPP} \subseteq \text{P}$. Theorem 3 scales smoothly over the entire derandomization spectrum for prRP. Due to losses in the generic conversion from hitting sets to derandomizations for two-sided error, the corresponding result for prBPP does not scale that well. In particular, a low-end variant of Theorem 3 for prBPP remains open. That said, the results are robust in a similar sense as above with respect to input lengths. In fact, the approach inherently yields a much higher degree of robustness because it effectuates a hardness vs. randomness tradeoff on an input-by-input basis, as we explain further in the paragraph below about our techniques.

As a summary of the above discussion, Table 1 provides a qualitative overview of the lower bound equivalences for each of the three types of derandomization considered.

Derandomization	Lower bound
blackbox, almost-all inputs	non-uniform
most inputs	uniform
whitebox, almost-all inputs	uniform, almost-all inputs

Table 1: Equivalences between various types of derandomization and lower bounds

In the setting of the third line in Table 1, a full-fledged equivalence along the lines of Chen and Tell remains open due to the additional uniform-circuit depth requirement that is needed in the direction from hardness to derandomization. As such, we refer to their results as *near-equivalences.*

Later works managed to obtain full-fledged equivalences with other hardness conditions, all related to compression. Liu and Pass did so for hardness of separating high from low Levin-Kolmogorov complexity [LP22] as well as for hardness in the presence of efficiently-computable leakage [LP23]. Korten [Kor22] established an equivalence with the existence of a deterministic polynomial-time algorithm for the following problem: Given a probabilistic circuit $C_{\text{comp}} : \{0,1\}^n \to \{0,1\}^{n-1}$ and a deterministic circuit $C_{\text{dec}} : \{0,1\}^{n-1} \to \{0,1\}^n$, find a string $z \in \{0,1\}^n$ such that $C_{\text{dec}}(C_{\text{comp}}(z))$ differs from z with high probability. Chen, Tell, and Williams [CTW23] viewed such an algorithm as a refuter for the identity function against a class $\mathcal{A}$ of algorithms that go through a compression phase, reduced the class $\mathcal{A}$, and extended the result to efficiently computable multi-bit functions other than identity. Their framework also captures the equivalences from [LP22] and [LP23].

Theorem 4 ([CTW23]). *The following are equivalent:*

1. *For some constant $\epsilon \in (0,1)$, there exists a polynomial-time list-refuter for the identity function against* $\text{prBPTICOMP}[n^{1+\epsilon}, n^{\epsilon}]$.
2. *For some constants $a \geq 1$ and $\epsilon \in (0,1)$, there exists a function computable in deterministic time n^a that admits a deterministic polynomial-time list-refuter against the class* $\text{prBPTICOMP}[n^{a+\epsilon}, n^{\epsilon}]$.
3. $\text{prBPP} \subseteq \text{P}$.

For a class $\mathcal{A}$ of algorithms, $\mathcal{A}\text{TICOMP}[t(n), s(n)]$ denotes the class of computational processes obtained by first running a probabilistic algorithm A_{comp} and then an algorithm $A_{\text{dec}} \in \mathcal{A}$ on the output of A_{comp} such that both A_{comp} and A_{dec} run in time $t(n)$ and A_{comp} outputs a string of length at most $s(n)$. Assuming $s(n) < n$, one can view A_{comp} as producing a compressed representation of the input, from which A_{dec} computes the output. We refer to such pairs of algorithms as *bottleneck* algorithms. A *refuter* for a function f against a class $\mathcal{A}'$ is a meta algorithm that, given as input the description of an algorithm $A' \in \mathcal{A}'$ and a length n, finds an input z of length at least n on which A' fails to compute f. A *list-refuter* similarly outputs a list $z_1, \ldots, z_\tau$ of inputs of length at least n that contains at least one z_i on which A' fails to compute f.

Note that the existence of the refuter in Theorem 4 only guarantees that, for any fixed $A' = (A_{\text{comp}}, A_{\text{dec}})$ in $\text{prBPTICOMP}[t(n)^{1+\epsilon}, n^{\epsilon}]$ there exist infinitely many inputs on which A' fails to compute f. This stands in contrast with Theorem 3, where the algorithm is required to fail on almost-all inputs. However, in the refutation setting the counterexamples need to be found efficiently. In the case of hardness on almost-all inputs, there are trivial refuters, e.g., output 0^n for length n.

AM setting. An equivalence corresponding to the first line of Table 1 is known throughout the entire spectrum [KvM02, MV05, SU05]. The role of EXP is now taken over by $\text{NEXP} \cap \text{coNEXP}$, and the circuits are nondeterministic (or single-valued nondeterministic, or deterministic with oracle access to an NP-complete problem like SAT). The simulations use hitting-set generators for AM that are efficiently computable nondeterministically. Hitting-set generators are the natural constructs in the setting of AM because every Arthur-Merlin protocol can be efficiently transformed into an equivalent one with perfect completeness. As in the BPP setting, the lower bound equivalences for blackbox derandomization of prAM scale smoothly and are robust with respect to input lengths.

Regarding derandomizations that work on all but a negligible fraction of the inputs of a given length (the second line in Table 1), no hardness vs. randomness tradeoffs for AM were known prior to our work. What was known, are high-end results on derandomizations where no efficient nondeterministic algorithm can locate inputs on which the simulation is guaranteed to be incorrect [GSTS03, SU09]. Indeed, the authors of [GSTS03] explicitly mention the average-case setting and why their approach fails to yield average-case simulations that are correct on a large fraction of the inputs. The setting corresponding to the third line in Table 1 was not studied before.

Main results. As our main results, we obtain near-equivalences in this third setting, i.e., between whitebox derandomizations of Arthur-Merlin protocols that work on almost-all inputs, on the one hand, and hardness on almost-all inputs against Arthur-Merlin protocols, on the other hand.

We start from a similar observation in the derandomization to hardness direction as the one Chen and Tell made for BPP at the high end of the spectrum. We refer to Section 5.1 for the quantification of "a few".

Proposition 5. *If* prAM $\subseteq$ NP*, then for every constant c there exists a length-preserving function f that is computable in nondeterministic polynomial time with "a few" bits of advice, and is hard on almost-all inputs against* AMTIME$[n^c]$.

Importantly, we are able to establish an almost-converse of Proposition 5. Under a slightly stronger hardness assumption, we construct a targeted hitting-set generator for prAM that is computable in nondeterministic polynomial time, yielding the following derandomization result.

Theorem 6. *Let f be a length-preserving function computable in nondeterministic time n^a for some constant a. If f is hard on almost-all inputs against* prAMTIME$[n^c]$ *for $c = O((\log a)^2)$, where $O(\cdot)$ hides some universal constant, then there exists a targeted hitting-set generator that achieves the derandomization*

$$\text{prAM} \subseteq \text{NP}.$$

In contrast to Theorem 3 in the BPP setting, Theorem 6 in the AM setting has no uniform-circuit depth restriction on the function f. Together with Proposition 5, Theorem 6 represents a near-equivalence between prAM $\subseteq$ NP and hardness on almost-all inputs of length-preserving functions against Arthur-Merlin protocols. Whereas in the BPP setting, the remaining gap relates to uniform-circuit depth, in the AM setting the remaining gap relates to the advice and the technical distinction between AM and prAM protocols. Note that the focus on length-preserving functions f in Proposition 5 and Theorem 6 is for concreteness. For Proposition 5 to hold, the number of output bits needs to grow with n in an efficiently computable fashion. For Theorem 6 any number of output bits suffices as long as there are not so many that the function f becomes trivially hard for Arthur-Merlin protocols running in time n^c.

Both Proposition 5 and Theorem 6 scale quite smoothly across the derandomization spectrum. The generalization of Theorem 6 has the following form: Let f be a length-preserving function computable in nondeterministic time $T(n)$. If f is hard on almost-all inputs against prAMTIME$[t(n)]$, then prAM $\subseteq$ NTIME[poly$(T(n))$]. Intuitively, we may think of $t(n)$ as only slightly smaller than $T(n)$ for high-end results and much smaller for low-end results. Pushing our techniques as far as possible toward the low end, we obtain the following variant of Theorem 6.

Theorem 7. *Let f be a length-preserving function computable in nondeterministic exponential time. If f is hard on almost-all inputs against* prAMTIME$[n^{b(\log n)^2}]$ *for all constants b, then for some constant c there exists a targeted hitting-set generator that achieves the derandomization*

$$\text{prAM} \subseteq \text{NTIME}[2^{n^c}]. \qquad (1)$$

As prAM $\subseteq$ NEXP trivially holds, the conclusion (1) of Theorem 7 represents the very low end of the derandomization spectrum. Note that a perfectly smooth scaling of Theorem 6 would only need a polynomial lower bound to arrive at the conclusion of Theorem 7, but the hypothesis of Theorem 7 requires a lower bound of $n^{\omega((\log n)^2)}$. We remark that the same discrepancy shows up in the current best-scaling uniform hardness vs. randomness tradeoffs for AM [SU09]. We refer to Theorem 32 in Section 4 for the full scaling and to Table 2 in the same section for other interesting instantiations.

We also develop a *full* equivalence in the *refutation* setting, where the refuter is supposed to produce counterexamples for Arthur-Merlin protocols that go through a compression phase, i.e., *bottleneck* protocols. Let $(A_{\text{comp}}, P_{\text{dec}})$ be in prAMTICOMP$[t(n), s(n)]$. Note that A_{comp} is still a probabilistic algorithm, while P_{dec} is a promise Arthur-Merlin protocol. We say that $(A_{\text{comp}}, P_{\text{dec}})$ is *sound* for a function f if for all inputs z, with high probability, $P_{\text{dec}}(A_{\text{comp}}(z))$ either correctly computes $f(z)$ or else indicates failure.

We show that targeted hitting-set generators that suffice to derandomize prAM are equivalent to nondeterministic refuters for identity against bottleneck Arthur-Merlin protocols that are guaranteed to be sound for identity, and that identity can be replaced by an existentially quantified function f computable in nondeterministic polynomial time.

Theorem 8. *The following are equivalent:*

1. *For some constant* $\epsilon \in (0, 1)$*, there exists a nondeterministic polynomial-time list-refuter for the identity function against* prAMTICOMP$[n^{1+\epsilon}, n^{\epsilon}]$ *protocols with promised soundness for identity.*

2. *For some constants* $a \geq 1$ *and* $\epsilon \in (0, 1)$*, there exists a function* f *computable in nondeterministic time* n^a *that admits a nondeterministic polynomial-time list-refuter against* prAMTICOMP$[n^{a+\epsilon}, n^{\epsilon}]$ *protocols with promised soundness for* f*.*

3. *There exists a targeted hitting-set generator that achieves the derandomization* prAM $\subseteq$ NP*.*

Consider the first statement in Theorem 8. Because of the bottleneck, any fixed protocol $(A_{\text{comp}}, P_{\text{dec}})$ of the stated type fails to compute identity on a random input of sufficiently large length. Thus, the identity function admits a trivial refuter meeting the requirements of the theorem except that the refuter is *probabilistic* instead of deterministic. From this perspective, Theorem 8 shows that for derandomizing prAM, it suffices to derandomize trivial refuters for the identity function.

Theorem 8 scales smoothly in terms of the running time for the refuter. A refuter for the function f that runs in time T results in a targeted hitting-set generator that runs in time poly$(T(\text{poly}(n)))$. Similarly, a targeted hitting-set generator that runs in time T, and thus achieves the derandomization prAM $\subseteq$ NTIME$[T(\text{poly}(n))]$, results in a refuter for identity that runs in time $T(\text{poly}(n))$. When the running time of the refuter ranges from polynomial to subexponential, so does the time needed for the nondeterministic simulations, covering the entire derandomization spectrum.

Byproducts. Using our targeted hitting-set generators we are able to make progress on a number of related topics. We mention three representative ones here; more are described in the body of the paper.

First, there is the relationship between whitebox derandomization of prAM and the existence of targeted hitting-set generators for prAM. In the paper [Gol11] where Goldreich introduced targeted pseudo-random generators for prBPP and showed that their existence is equivalent to whitebox derandomization of prBPP, he asked about analogous results for prAM. To the best of our knowledge, there have been no prior results along those lines. We take a first step toward an equivalence in this setting.

Theorem 9. *If* $\mathrm{prAMTIME}[2^{\mathrm{polylog}(n)}] \subseteq \text{io-NEXP}$*, then there exists a targeted hitting-set generator for* prAM *that yields the simulation* $\mathrm{prAM} \subseteq \text{io-NTIME}[2^{n^c}]/n^\epsilon$ *for some constant c and all* $\epsilon > 0$.

Second, we establish the first hardness vs. randomness tradeoffs for Arthur-Merlin protocols in the average-case setting. Informally, under a high-end worst-case hardness assumption, we obtain nondeterministic polynomial-time simulations of prAM that are correct on all but a negligible fraction of the inputs.

Theorem 10. *If* $\mathrm{NTIME}[2^{an}] \cap \mathrm{coNTIME}[2^{an}] \not\subseteq \mathrm{BPTIME}[2^{(\log(a+1))^2 n}]_{||}^{\mathrm{SAT}}$ *for some constant* $a > 0$*, then for every problem in* prAM *and all* $e > 0$ *there exists a simulation of the problem in* NP *that is correct on all but a fraction* $1/n^e$ *of the inputs of length n for infinitely many lengths n.*

The class $\mathrm{BPTIME}[t(n)]_{||}^{\mathrm{SAT}}$ denotes probabilistic algorithms with bounded error that run in time $t(n)$ and can make parallel (i.e., non-adaptive) queries to an oracle for SAT. Theorem 10 answers a question in [GSTS03], which presents results in the different but related "pseudo" setting, where the simulation may err on many inputs of any given length, but no polynomial-time nondeterministic algorithm can pinpoint an error at that length. We remark that our technique also leads to identical results in the "pseudo" setting by replacing the hardness assumption with hardness against $\mathrm{AMTIME}[t(n)]$.

The model $\mathrm{prBPP}_{||}^{\mathrm{SAT}}$ was used as a proxy for prAM in the initial derandomization results for Arthur-Merlin protocols [KvM02] and is seemingly more powerful. However, derandomization results for prAM typically translate into similar derandomization results for $\mathrm{prBPP}_{||}^{\mathrm{SAT}}$. In particular, the conclusion $\mathrm{prAM} \subseteq \mathrm{NP}$ of Theorem 6 implies that $\mathrm{prBPP}_{||}^{\mathrm{SAT}} \subseteq \mathrm{P}_{||}^{\mathrm{SAT}}$, and the conclusion $\mathrm{prAM} \subseteq \mathrm{NTIME}[2^{n^c}]$ for some constant c in Theorem 7 implies that $\mathrm{prBPP}_{||}^{\mathrm{SAT}} \subseteq \mathrm{DTIME}[2^{n^c}]_{||}^{\mathrm{SAT}}$ for some constant c. In the case of Theorem 10, we argue that the hardness assumption implies simulations of $\mathrm{prBPP}_{||}^{\mathrm{SAT}}$ in $\mathrm{P}_{||}^{\mathrm{SAT}}$ of the same strength as the simulations of prAM in NP. This way, we obtain a hardness vs. randomness tradeoff in which the hardness model and the model to-be-derandomized match, namely probabilistic algorithms with bounded error and non-adaptive access to an oracle for SAT.

As our third byproduct, we present an unconditional mild derandomization result for AM in the average-case setting. By a *mild* derandomization of AM we mean a nontrivial simulation on Σ_2-machines. Recall that $\mathrm{AM} \subseteq \Pi_2\mathrm{P}$, and proving that $\mathrm{AM} \subseteq \Sigma_2\mathrm{P}$ is a required step if we hope to show that $\mathrm{AM} \subseteq \mathrm{NP}$. It is known that AM can be simulated (at infinitely many input lengths n) on Σ_2-machines that run in subexponential time and take n^c bits of advice for some constant c [Wil16]. It remains open whether AM can be simulated on Σ_2-machines in subexponential time with subpolynomial advice. Indeed, such a simulation for prAM would imply lower bounds against nondeterministic circuits that are still open [AvM17]. We show an unconditional subexponential-time and subpolynomial-advice Σ_2-simulation for prAM in the average-case setting.

Theorem 11. *For every problem in* prAM *and every constant* $\epsilon > 0$ *there exists a simulation of the problem in* $\Sigma_2\mathrm{TIME}[2^{n^\epsilon}]/n^\epsilon$ *that is correct on all but a fraction* $1/n^e$ *of the inputs of length n, for all constants e and infinitely many lengths n.*

In fact, we can extend Theorem 11 to $\mathrm{prBPP}_{||}^{\mathrm{SAT}}$ in lieu of prAM.

Techniques. For our main results, we develop an instance-wise transformation of hardness into targeted hitting sets tailored for AM. In the setting of BPP, Chen and Tell combine the Nisan-Wigderson pseudo-random generator construction [NW94] with the doubly-efficient proof systems of Goldwasser, Kalai, and Rothblum [GKR15] (as simplified in [Gol18]). The latter allows them to capture the computation of a uniform circuit of size T and depth d for f on a given input x by a downward self-reducible sequence of polynomials, which they use to instantiate the NW generator. In case the derandomization of a one-sided error algorithm on a given input x fails, a bootstrapping strategy à la [IW01], based on a learning property of the NW generator, allows them to retrieve the value of $f(x)$ in time $O(d \cdot \mathrm{polylog}(T))$. Thus, provided the depth d is small compared to the size T, either the derandomization on input x works or else the computation of $f(x)$ can be sped up. In the refutation setting, the refuter provides an input z. The bootstrapping strategy produces a small circuit that computes the mapping $i \mapsto f(z)_i$; the compression algorithm A_{comp} outputs this circuit; the algorithm A_{dec} takes the circuit and evaluates it on all i to determine and output $f(z)$.

A similar approach based on [GKR15] applies to the AM setting by replacing the NW construction with a hitting-set generator construction for AM that also has the learning property. Like in the BPP setting, the construction is of more interest when the circuits for f have relatively small depth. Moreover, the construction can only handle a limited amount of nondeterminism in the computation for f, whereas the direction from derandomization to hardness seems to require more.

In order to remedy both shortcomings, we develop a new method to extract hardness from a nondeterministic computation on a given input z, based on probabilistically checkable proofs of proximity (PCPPs) rather than [GKR15]. The soundness of our method presupposes some type of resilience of the underlying regular pseudo-random generator. The required property was first identified and used by Gutfreund, Shaltiel and Ta-Shma [GSTS03] for the Miltersen-Vinodchandran generator MV [MV05], and later by Shaltiel and Umans [SU09] for their recursive variant of the MV generator, RMV. We combine RMV with the probabilistically checkable proofs of proximity of Ben-Sasson, Goldreich, Harsha, Sudan, and Vadhan [BSGH$^+$05] to transform hardness into pseudo-randomness for AM in an instance-wise fashion, without any uniform-circuit depth restriction or limitation on the amount of nondeterminism.

We highlight one strong feature of *all* instance-wise approaches. If the hardness condition holds on almost-all inputs, then the derandomization works on almost-all inputs. This is the setting in which we stated the results of Chen and Tell and our main results. Similarly, if the hardness condition holds on all inputs of a given length, then the derandomization works on all inputs of that length. This is the robustness property that we alluded to earlier. However, an instance-wise approach yields much more, including average-case derandomization results: To obtain a nondeterministic simulation for some prAM problem that works with high probability over any given distribution, it suffices to assume that every prAM protocol can only compute the hard function f with low probability over that same distribution. In the refutation setting, we have the refuter provide the input z, and we show that our reconstructor is actually a bottleneck Arthur-Merlin protocol.

Our derandomization-to-hardness result follows by diagonalization, as does the one by Chen and Tell. The result that targeted generators sufficient for derandomizing prAM imply refutation works by employing such generators to derandomize the process of obtaining a counterexample at random. The resilience property of our generator that follows from our use of PCPPs and the resilience property of the RMV reconstructor play a critical role here.

To obtain our byproducts, we combine our targeted hitting-set generator with several other ingredients, including diagonalization, the "easy-witness" method and traditional hardness vs. randomness tradeoffs. Our average-case derandomization results require a modification of our targeted HSG so that it respects a stronger resilience property. Along the way to our unconditional mild derandomization result, we establish an "easy witness lemma" for Σ_2 computations, which may be of independent interest.

Organization. In Section 2, we develop the ideas behind our results and relate them to existing techniques. We start the formal treatment in Section 3 with definitions, notation, and other preliminaries. In Section 4, we construct our targeted HSG and establish our hardness-to-derandomization results that make use of it (Theorems 6, 7 and the refutation-to-derandomization direction of Theorem 8). Section 5 presents the derandomization-to-hardness side of our near-equivalence, the targeted-generators-to-refutation side of Theorem 8, and a proof of our byproduct on derandomization to targeted hitting-set generators (Theorem 9). In Section 6, we derive our derandomization byproducts under uniform worst-case hardness (the average-case simulation of Theorem 10 as well as a simulation that works on all inputs of infinitely many lengths). Section 7 contains our unconditional mild derandomization result for AM (Theorem 11).

2 Technical overview

In this section, we start with an overview of techniques used in prior hardness vs. randomness tradeoffs for BPP and AM in a way that facilitates a high-level exposition of our main hardness-to-derandomization result for AM. We also provide the intuition for our derandomization-to-hardness result and for our byproducts.

2.1 Main results

We start with an overview of the techniques used for hardness-to-derandomization results in the traditional setting for BPP (lines 1 and 2 in Table 1), followed by those in the new setting (line 3 in Table 1). We then transition to AM, discuss the additional challenges, the known techniques in the traditional setting and, finally, our results in the new setting.

Traditional setting for BPP. The key ingredient in all known hardness vs. randomness tradeoffs is a pseudo-random generator construction G that takes a function h as an oracle and produces a pseudo-random distribution G^h with the following property: Any statistical test D that distinguishes G^h from uniform suffices as an oracle to efficiently learn h approximately from a small number of queries. Thus, if G^h does not "look random" to an efficient randomized process A on an input x, an approximation to h can be reconstructed efficiently when provided with x and the values of h on a small number of points, as well as oracle access to the distinguisher $D(r) = A(x, r)$, where $A(x, r)$ denotes the output of A on input x and random-bit string r. If the function h can be self-corrected (e.g., by being random self-reducible or by its truth table being a codeword in a locally-correctable error-correcting code), then the exact function h can be reconstructed efficiently.

In order to obtain hardness vs. randomness tradeoffs from pseudo-random generator constructions with the learning property, two questions need to be addressed:

1. How to obtain the distinguishers D?
2. How to obtain the answers to the learning queries?

The first question asks how to find inputs x on which the process A is not fooled by G^h. In the non-uniform setting such an input can be included in the advice. In the uniform setting for BPP, such inputs can be found by sampling x at random and testing for a difference in behavior of $D \doteq A(x,\cdot)$ between the uniform and the pseudo-random distributions, which can be done in prBPP.

Regarding the second question, in the non-uniform setting, the answers to the learning queries can also be provided as advice. In the uniform setting, [IW01] employs a function h that is not only random self-reducible but also downward self-reducible, and uses the downward self-reduction to answer the learning queries for length n by evaluating the circuit that resulted from the reconstruction for length $n-1$. This bootstrapping strategy presupposes that the reconstruction works at almost-all input lengths. This is why we only know how to obtain simulations that are correct at infinitely many input lengths in the uniform setting for BPP.

New setting for BPP. In the setting of line 3 in Table 1, the role of pseudo-random generators is taken over by *targeted* pseudo-random generators. Whereas PRGs are oblivious to x (beyond its length), targeted PRGs take x as an input and are only supposed to fool the randomized process on that particular x. This approach obviates the problem of obtaining the distinguisher D (question 1 above) as we can use $D = A(x,\cdot)$ for the given x. Indeed, an equivalent formulation of targeted generators considers D itself as the input, and the targeted generator only needs to "fool" that particular D [Gol20]. Targeted PRGs can be constructed from a PRG G by instantiating G with an oracle $h = h_x$ that depends on x. This raises a third question in the application of a PRG for hardness vs. randomness tradeoffs:

3. How to obtain the function h_x from x?

Chen and Tell [CT21] use the doubly-efficient proof systems of Goldwasser, Kalai, and Rothblum [GKR15] (as simplified in [Gol18]) to obtain h_x from x and combine it with the Nisan-Wigderson pseudo-random generator construction [NW94]. The GKR proof system takes a logspace-uniform family of circuits of size $T(n)$ and depth $d(n)$ computing a (multi-bit) Boolean function f, and transforms the circuit for f on a given input x into a downward self-reducible sequence of multivariate low-degree polynomials $\hat{g}_{x,0} \ldots, \hat{g}_{x,d'(n)}$ where $d'(n) = O(d(n)\log(T(n)))$. The polynomial $\hat{g}_{x,0}$ is efficiently computable at any point given input x, and the value of $f(x)$ can be extracted efficiently from $\hat{g}_{x,d'(n)}$. We refer to the sequence of polynomials as a *layered arithmetization* of the circuit for f on input x.

Chen and Tell instantiate the NW generator with the Hadamard encoding of each of the polynomials $\hat{g}_{x,i}$ as the function $h = h_{x,i}$, and follow a bootstrapping strategy similar to [IW01] to construct $\hat{g}_{x,d'(n)}$ from $\hat{g}_{x,0}$. For the strategy to work, the NW reconstructor needs to succeed at every level. This is the reason why Chen and Tell only end up with a (targeted) *hitting-set* generator rather than a *pseudo-random* generator. The time required by the bootstrapping process is proportional to the number of layers and thus to the depth $d(n)$ of the circuit computing f. By setting the parameters of the arithmetization appropriately, the dependency on the size $T(n)$ is only polylogarithmic. This is what enables the reconstruction to compute $f(x)$ very quickly as long as the depth $d(n)$ is not too large.

The above approach hinges on the speed of the reconstruction process. Subsequent works hinge on the compressed representation that the reconstruction process implicitly builds, which can be viewed as a bottleneck that the computation goes through. Such approaches typically allow for a matching implication from derandomization to hardness because a random function cannot be compressed and derandomization lets us find such an incompressible function deterministically.

Like Chen and Tell, Liu and Pass also apply the NW generator but obtain h_x differently. In [LP22], they use h_x that are the encodings of the outputs $\pi(x)$ of all small efficient programs π (so the resulting string has small Kolmogorov-Levin complexity Kt). The answers to the learning queries are hard-wired into the program that reconstructs h_x. The direction from derandomization to hardness follows from the fact that an efficient algorithm that separates low from high Kolmogorov-Levin complexity acts as a distinguisher. In [LP23], h_x encodes the value of $f(x)$ itself, where f is an almost-all inputs leakage-resilient hard function (a function that remains hard even if some efficiently-computable information about $f(x)$ is leaked to an attacker). The approach leads to a (targeted) *pseudo-random* generator as it only involves a single h_x. The answers to the learning queries are provided as part of the information about $f(x)$ that is leaked, and the direction from derandomization to hardness follows the typical pattern.

Recall that the reconstructor only needs access to D and the answers to the learning queries to h_x; let $A_{\text{rec}}(h_x, D)$ denote the result. Each of the above approaches can be viewed as an explicit construction of one or more h_x from x such that

$$A_{\text{rec}}(h_x, D) \neq h_x \tag{2}$$

for at least one h_x. This suffices because (2) means that the reconstruction fails for h_x, and whenever that happens the targeted pseudo-random generator based on h_x has to fool D. Prior approaches all guarantee (2) indirectly by constructing the functions h_x out of a function f with a particular hardness property, and showing that if all h_x satisfy $A_{\text{rec}}(h_x, D) = h_x$, then the hardness property for f on input x fails. Prior approaches are also oblivious to $D \doteq A(x, \cdot)$ but that feature is nothing special as one can always incorporate a description of A as part of the input x.

Recent approaches take a broader perspective and try to directly construct h_x with the sole requirement that (2) holds. Thanks to the bottleneck that the reconstruction process goes through, we know that a random choice of h_x satisfies the requirement. Under the derandomization hypothesis prBPP $\subseteq$ P, we can efficiently find such an h_x *deterministically.* Conversely, if we can efficiently find such an h_x deterministically, we obtain an efficient targeted pseudo-random generator in the BPP setting.

Korten [Kor22] follows this outline, where the circuit C_{comp} computes the compressed representation of a candidate value z for h_x based on D, from which the circuit C_{dec} attempts to retrieve h_x. Korten does not use the full NW construction but only Yao's predictor, thereby only achieving a modest compression. Chen, Tell, and Williams [CTW23] achieve better compression using the full NW construction. They also cast the construction of h_x as a refuter for the identity function $f(z) = z$ against the reconstructor algorithm $A_{\text{rec}}(z, D)$, and show how the identity function can be replaced by any efficiently computable length-preserving function f. The extension sets $h_x = f(z)$ and involves an application of the Chen-Tell bootstrapping approach (based on the standard circuit simulation of the uniform computation of f) in order to obtain the answers to the learning queries. As a consequence, the targeted generator is only hitting. In the special case of identity, the learning queries are simply bits of z, which obviates the need for Chen-Tell and results in a targeted generator that is pseudo-random.

Transition to AM. A number of changes are in order in terms of the requirements for similar results for AM. First, we need to handle *co-nondeterministic* distinguisher circuits D instead of deterministic ones. Co-nondeterministic circuits suffice because Arthur-Merlin protocols can be assumed to have perfect completeness. The only requirement for a correct derandomization is in the case of negative instances, in which case we want to hit the set of Arthur's random-bit strings for which Merlin cannot produce a witness. By the soundness property of the Arthur-Merlin protocol, the set contains at least half of the random-bit strings.

Second, we need to accommodate *nondeterministic* algorithms computing the function f. This is because the direction from derandomization to hardness seems to need them (see Proposition 5). On each input z, such an algorithm needs to have at least one successful computation path, and on every successful computation path, the output should equal $f(z)$. Similarly, we allow for nondeterministic refuters, which need to output counterexamples on every accepting computation path.

Third, the algorithm for the targeted hitting-set generator can also be nondeterministic, which is natural when the algorithm or refuter for f is nondeterministic. In the case of a generator, the nondeterministic algorithm should still have at least one successful computation path on every input, but it is fine to produce different outputs on different successful computation paths. For any given D, on every successful computation path, the output should be a hitting set for D. This allows us to nondeterministically simulate a promise Arthur-Merlin protocol P on input x as follows: First, construct the circuit D based on P and x and guess a computation path of the targeted HSG on input D; if it succeeds, say with output S, guess a computation path for P on input x using each of the elements in S as the random-bit string, and accept if all of them accept; otherwise, reject.

Finally, we need to be able to run the reconstruction procedure as a (promise) *Arthur-Merlin protocol*. This is because we want the model in which we can compute $f(z)$ in case of a failed derandomization, to match the class we are trying to derandomize. There are two requirements for the protocol to compute $f(z)$ on input z:

- *Completeness* demands that there exists a strategy for Merlin that leads Arthur to succeed with output $f(z)$ with high probability.
- *Soundness* requires that, no matter what strategy Merlin uses, the probability for Arthur to succeed with an output other than $f(z)$ is small.

The reconstructor naturally needs the power of nondeterminism in order to simulate the distinguisher D. Making sure the reconstructor is sound and needs no more power than prAM is the challenge.

Traditional setting for AM. In reference to the first two questions above, the answer to the one about obtaining a distinguisher D is similar as for BPP, except that in the uniform setting we do not know how to check in prAM for a difference in behavior of $D \doteq P(x, \cdot)$ between the uniform and the pseudo-random distributions. This is why average-case results remain open for AM. Instead, one assumes that some nondeterministic algorithm produces, on every successful computation path on input 1^n, an input x of length n on which the difference in behavior is guaranteed.

As for obtaining answers to the learning queries in the uniform setting for AM, we can make use of the nondeterminism allowed during the reconstruction and ask Merlin to provide the answers to the learning queries. However, we need to guard against a cheating Merlin. A strategy proposed by Gutfreund, Shaltiel and Ta-Shma in [GSTS03] consists of employing a function h that has a length-preserving instance checker. After Merlin has provided the supposed answers to the learning queries, to compute $h(z)$ for a given input z, we run the instance checker on input z and answer the queries y of the instance checker by running the evaluator part of the reconstruction process on input y. All the runs of the evaluator can be executed in parallel, ensuring a bounded number of rounds overall, which can be reduced to two in the standard way at the cost of a polynomial blowup in the running time [BM88].

To guarantee soundness, the reconstruction process needs to have an additional *resilience* property, namely that it remains partial single-valued even when the learning queries are answered incorrectly. Two hitting-set generators tailored for AM are known to have the property: the Miltersen-Vinodchandran generator MV [MV05], which is geared toward the high end, and a recursive version, RMV, developed by Shaltiel and Umans [SU09] to cover a broader range. MV is used for the high end in [GSTS03], and RMV for the rest of the spectrum in [SU09].

New setting for AM. A first approach to port the Chen-Tell result to the AM setting is to replace NW with a generator for AM that has the learning property and a reconstructor running in prAM. The nondeterminism allows one to run the bootstrapping process in parallel, so the number of rounds of Arthur and Merlin remains bounded, but the overall running time remains proportional to the depth of the circuits for f. This means that, like in the setting of BPP, this first approach only yields meaningful results when the depth is small compared to the size. Nondeterministic circuits for f can be accommodated in this approach by treating them as deterministic circuits with nondeterministic guess bits as additional inputs. However, this limits the amount of nondeterminism that can be handled. To address these issues, we develop a refined approach based on PCPs.

We build a targeted hitting-set generator for AM based on the RMV hitting-set generator. To obtain h_x from x in the setting of hardness on almost-all inputs, we make use of Probabilistically Checkable Proofs (PCPs) for the nondeterministic computation of the string $f(x)$ from x. Let V denote the verifier for such a PCP system that uses $O(\log(T(n))$ random bits and $\mathrm{polylog}(T(n))$ queries for nondeterministic computations that run in time $T(n)$. On input x, our targeted HSG guesses the value of $f(x)$ and a candidate PCP witness y_i for the i-th bit of $f(x)$ for each i, and runs all the checks of the verifier V on y_i (by cycling through all random-bit strings for V). If all checks pass, our targeted HSG instantiates RMV with y_i for each i as (the truth table of) the oracle h_x, and outputs the union of all the instantiations as the hitting set, provided those nondeterministic computations all accept; otherwise, the targeted HSG fails.

For the reconstruction of the i-th bit of $f(x)$, Arthur generates the learning queries of the RMV reconstructor for the oracle y_i, and Merlin provides the purported answers as well as the value of the i-th bit of $f(x)$. Arthur then runs some random checks of the verifier V on input x, answering the verifier queries by executing the evaluator of the RMV reconstructor. All the executions of the evaluator can be performed in parallel, ensuring a bounded number of rounds overall. The resilient partial single-valuedness property of the RMV reconstructor guarantees that the verifier queries are all consistent with some candidate proof $\tilde{y}_i$. The completeness and soundness of the PCP then imply the completeness and soundness of the reconstruction process for our targeted HSG. As V makes few queries and is very efficient, the running time of the process is dominated by the running time of the RMV reconstructor.

Abstracting out the details of our construction and how the distinguisher D is obtained, the result can be captured in two procedures: a nondeterministic one, H, which has at least one successful computation path for every input and plays the role of a targeted hitting-set generator, and a promise Arthur-Merlin protocol, P_{rec}, which plays the role of a reconstructor for the targeted hitting-set generator.

Property 12. *For every $z \in \{0,1\}^*$ and for every co-nondeterministic circuit D that accepts at least half of its inputs, at least one of the following holds:*

1. *$H(z, D)$ outputs a hitting set for D on every successful computation path.*
2. *$P_{rec}(z, D)$ computes $f(z)$ in a complete and sound fashion.*

Note that in Property 12 both H and P_{rec} are given access to the input z as well as the co-nondeterministic circuit D. In fact, the dependency of H on D is only through the number of input bits of D. For P_{rec}, blackbox access to D suffices (in addition to the input z). However, we may as well give both H and P_{rec} full access to the input z and the circuit D. In the setting of hardness on almost-all inputs, the co-nondeterministic circuit P_{rec} is obtained by hardwiring the input z into the Arthur-Merlin protocol being derandomized, but this is not essential for the construction.

Theorem 6 follows by considering nondeterministic running time $T(n) = n^a$ and co-nondeterministic circuits D of size n^c for some $c > 1$. In this regime, H runs in time $n^{O(a+c)}$ and P_{rec} in time $n^{O(c(\log a)^2)}$. Under the hypothesis of Theorem 6, the second item in Property 12 cannot happen except for finitely many z, so the first item needs to hold. For any constant $c' < c$, this yields a polynomial-time targeted hitting-set generator for prAMTIME$[n^{c'}]$, which can be used for all of prAM by padding. Theorem 7 follows along the same lines; the running time is dictated by the RMV reconstructor.

In the refutation setting we no longer need hardness to hold on almost-all inputs but instead need a meta-algorithm that finds inputs where a given bottleneck protocol fails. We again make use of Property 12 but now connect derandomization to refuters for the function f against bottleneck protocols. In the direction from refutation to derandomization, we use the refuter to find an input z for which the reconstructor fails (i.e., the second item in Property 12 does not hold). In that case, $H(z, D)$ must output a hitting set for D (the first item in Property 12 holds). A key property to ensure that the reconstructor behaves like a bottleneck protocol is that the RMV reconstructor yields a compressed representation of any h_x that fails as a basis for obtaining a hitting set. In our PCP-based construction, we used this property to compress PCPs for each bit of $f(z)$ to ultimately speed up the computation of $f(z)$. One complication in the refutation setting is that verifying PCPs requires full access to the input z, which seems to ruin the potential for compression. We resolve the complication by modifying the generator and additionally run RMV on z itself. This way, the reconstructor goes through a compressed representation of z from which it can efficiently recover z. We take the compressor A_{comp} to be the algorithm that, on input z, generates and answers the learning queries for z, producing the compressed representation of z. We then feed the compressed representation of z into P_{rec}, which uses the RMV evaluator to access z whenever that is necessary. With this approach, and starting from a function f computable in nondeterministic time n^a for some constant a, we can construct targeted HSGs that achieve the derandomization prAM $\subseteq$ NP from the existence of a refuter against bottleneck protocols with subpolynomial compression that run in time $n^{a+\epsilon} \cdot \text{poly}(n)$ for some $\epsilon > 0$, where the poly(n) term comes from the use of PCPs.

We can do better, and get rid of the multiplicative poly(n) term, by further refining the approach and employing probabilistically checkable proofs of proximity rather than PCPs. Given random access to the input z of length n and to a proof, a PCPP verifier runs in time polylog(n) instead of poly(n). PCPPs, however, are only sound when the input is far in relative distance from a true instance of the underlying decision problem, which makes them more suitable to inputs that are in error-correctable form. For this reason, we have the compressor A_{comp} first encode the input z with an error-correctable code that is computable in time $n \cdot \text{polylog}(n)$, and have P_{rec} employ the PCPP verifier with the encoded version. The RMV evaluator allows us to recover individual bits of z very efficiently, in particular in time that is sublinear in n, which can be absorbed in the $n^{a+\epsilon}$ term together with the running time for the PCPP verifier. This is how we obtain Theorem 8. Because of similar gains, we employ PCPPs in lieu of PCPs throughout the paper.

Derandomization to hardness. Proposition 5 is established by diagonalization. Under the prAM $\subseteq$ NP assumption, every fixed-polynomial time AM protocol computing a length-preserving function can be simulated in nondeterministic fixed-polynomial time. We would like to diagonalize against these simulating nondeterministic machines to construct our hard function. Due to the lack of an almost-everywhere hierarchy result for NTIME, we do not know how to do this efficiently for generic nondeterministic machines. This is where the advice comes to rescue: We use advice to indicate which nondeterministic machines are *single-valued* at a particular input length. We only need to consider single-valued machines, and diagonalizing against them is easy for a nondeterministic machine with a little more running time, but figuring out which nondeterministic machines are single-valued at a given input length is hard.

For the direction from derandomization to refutation in Theorem 8, assuming the existence of a targeted HSG sufficient for derandomizing prAM, the objective is to obtain a refuter for identity against polynomial-time bottleneck Arthur-Merlin protocols with subpolynomial compression bottlenecks. For any fixed such bottleneck protocol P_{rec}, a probabilistic argument guarantees that P_{rec} fails to compute identity for most strings z of length n. Moreover, our use of PCPPs together with the resilience property of the RMV reconstructor ensures that the reconstruction protocol P_{rec} always meets the soundness requirement. This means that a successful refuter provides an input z on which the completeness requirement fails. The latter property can be verified co-nondeterministically, which allows us to generate such a z using the presumed targeted HSG and thus obtain a refuter computable in nondeterministic polynomial time.

2.2 Byproducts

In this section, we develop the intuition for our byproducts.

Targeted hitting-set generators from derandomization (Theorem 9). To obtain a targeted HSG from derandomization of prAM, we employ our targeted hitting-set generator in a win-win argument. Either a complexity class separation holds, in which case a result of [IKW02] guarantees the existence of a regular (oblivious) hitting-set generator that yields the derandomization result, or we get a strong complexity class collapse. The collapse allows us to bypass some of the difficulties in diagonalizing against prAM protocols on almost-all inputs (one of the reasons we require advice in the derandomization-to-hardness direction of our near-equivalence), thus allowing us to do so efficiently and uniformly, and then instantiate our targeted hitting-set generator construction.

Average-case derandomization (Theorem 10). Our average-case derandomization results under worst-case hardness assumptions also make use of our targeted hitting-set generator construction, but in a different way. They do not exploit the potential of the hitting sets to depend on the input x. In fact, they set $f(x)$ to the truth table of the worst-case hard language L from the hypothesis at an input length determined by $|x|$. Instead, they hinge on the strong resilient soundness properties of the reconstructor.

As we are considering the average-case derandomization setting, the problem of obtaining the distinguisher D for the reconstruction resurfaces. Our approach is similar to the one for the traditional average-case derandomization setting for BPP. If the simulation fails for protocol P with noticeable probability over a random input, then we can sample multiple inputs $x_1, x_2, \ldots$ and construct a list of "candidate distinguishers" $D_{x_1} \doteq P(x_1, \cdot), D_{x_2} \doteq P(x_2, \cdot), \ldots$ such that the list contains, with high probability, at least one "true" distinguisher. Whereas in the BPP setting one can test each candidate and discard, with high probability, the ones that are not distinguishers, we do not know how to do that in the AM setting. Instead, we employ a different approach: We run the reconstructor with each distinguisher with the hope that every execution either fails or outputs the correct value.

This approach necessitates a stronger form of resilience than the one provided by the RMV generator: That its reconstruction is sound when given as input *any* co-nondeterministic circuit D, not just those that accept at least half of their inputs (as in Property 12). We don't know how to guarantee this with our prAM reconstruction, but we are able to do so in $\mathrm{prBPP}_{||}^{\mathrm{SAT}}$ by approximating the fraction of inputs that D accepts and outright failing if the fraction is too low.

We point out that earlier works [GSTS03, SU09] also manage to guarantee soundness of the reconstructor for co-nondeterministic circuits D that accept at least half of their inputs, based on the resilient partial single-valuedness of the reconstructor for MV or RMV. They do so by running an instance checker, which limits the hard function f to classes for which instance checkers are known to exist, such as complete problems for E and EXP. Instead, we achieve soundness of the reconstructor based on the soundness of a PCPP. As PCPPs exist for all nondeterministic computations, this makes our approach more suitable in this setting. In particular, we do not know how to obtain Theorem 10 along the lines of [GSTS03, SU09].

Unconditional mild derandomization (Theorem 11). Our unconditional mild derandomization result relies on a similar win-win argument as in the proof of Theorem 9: Either some hardness assumption/class separation holds, in which case we get derandomization right away, or we get a complexity collapse that we use to construct, by diagonalization, a hard function f that has the efficiency requirements we need to obtain the derandomization result using our targeted hitting-set generator.

Since our result is unconditional, we cannot use derandomization assumptions to make diagonalizing against prAM protocols easier. Instead, we rely on the inclusion $\mathrm{prAM} \subseteq \Pi_2\mathrm{P}$, which allows for diagonalizing against such protocols in $\Sigma_2\mathrm{TIME}[n^{\omega(1)}]$. Our generator, however, requires the hard function to be computable by efficient nondeterministic algorithms. To help bridge the gap, we prove an "easy witness lemma" for Σ_2 computations that guarantees a strong collapse in case the aforementioned hardness assumption does not hold. The collapse then allows us to instantiate our targeted hitting-set generator construction with the diagonalizing function.

3 Preliminaries

We assume familiarity with standard complexity classes such as NP, AM, and prAM. We often consider inputs and outputs from non-Boolean domains, such as $\mathbb{F}^r$ for a field $\mathbb{F}$ and $r \in \mathbb{N}$. In such cases, we implicitly assume an efficient binary encoding for the elements of these domains. Finally, as is customary, all time bounds t are implicitly assumed to be time-constructible and satisfy $t(n) \geq n$.

3.1 Nondeterministic, co-nondeterministic and single-valued computation

We make use of nondeterministic, co-nondeterministic, and single-valued circuits in our results. A nondeterministic circuit is a Boolean circuit C with two sets of inputs, x and y. We say that C accepts x if there exists some y such that $C(x, y) = 1$, and that C rejects x otherwise. A co-nondeterministic circuit has a symmetric acceptance criterion: It accepts x if for all y it holds that $C(x, y) = 1$, and rejects x otherwise. A partial single-valued circuit also has two inputs, x and y; on input (x, y) it either fails (which we represent by $C(x, y) = \perp$) or succeeds and outputs a bit $b = C(x, y)$. Moreover, we require that for all y, y' such that both $C(x, y)$ and $C(x, y')$ succeed, $C(x, y) = C(x, y')$, i.e., the circuit computes a partial function on its first input. If, furthermore, for all x there exists a y such that $C(x, y)$ succeeds, we call the circuit total single-valued or just single-valued.

We are also interested in nondeterministic algorithms that compute total relations $R \subseteq \{0, 1\}^* \to \{0, 1\}^*$. Let T be a time bound. We say that a nondeterministic algorithm N computes R if for all $x \in \{0, 1\}^*$, there exists at least one computation path on which $N(x)$ succeeds, and $N(x)$ outputs some $y \in R(x)$ on all successful computation paths. Note, in particular, that if a function $f : \{0, 1\}^* \to \{0, 1\}^*$ is computable in nondeterministic time $T(n)$, then the language $L_f = \{(x, i, b) \mid f(x)_i = b\}$ is in NTIME$[T(n)]$.

3.2 Arthur-Merlin protocols

A promise Arthur-Merlin protocol P is a computational process in which Arthur and Merlin receive a common input x and operate as follows in alternate rounds for a bounded number of rounds. Arthur selects a random string and sends it to Merlin. Merlin sends a string that depends on the input x and all prior communication from Arthur; the underlying function is referred to as Merlin's strategy, which is computationally unrestricted. At the end of the process, a deterministic computation on the input x and all communication determines acceptance. The running time of the process is the running time of the final deterministic computation.

Any promise Arthur-Merlin protocol can be transformed into an equivalent one with just two rounds and Arthur going first, at the cost of a polynomial blow-up in running time, where the degree of the polynomial depends on the number of rounds [BM88]. As such, we often use the notation prAM to refer to promise Arthur-Merlin protocols with any bounded number of rounds, even though, strictly speaking, the notation refers to a two-round protocol with Arthur going first.

Promise Arthur-Merlin protocols can be simulated by probabilistic algorithms with oracle access to SAT: Instead of interacting with Merlin, Arthur asks the SAT oracle whether there exists a response of Merlin that would lead to acceptance. Similarly, $\mathrm{P}_{||}^{\mathrm{prAM}}$ can be simulated in $\mathrm{BPP}_{||}^{\mathrm{SAT}}$, the class of problems decidable by probabilistic polynomial-time algorithms with bounded error and non-adaptive oracle access to SAT. In fact, a converse also holds and helps to extend some of our results for prAM to the class $\mathrm{prBPP}_{||}^{\mathrm{SAT}}$.

Lemma 13 ([CR11]). $\mathrm{prBPP}_{||}^{\mathrm{SAT}} \subseteq \mathrm{P}_{||}^{\mathrm{prAM}}$.

In Lemma 13, the deterministic machines with oracle access to prAM on the right-hand side are guaranteed to work correctly irrespective of how the queries outside of the promise are answered, even if those queries are answered inconsistently, i.e., different answers may be given when the same query is made multiple times.

Arthur-Merlin protocols that output values. A promise Arthur-Merlin protocol P may also output a value. In this case, at the end of the interaction, the deterministic computation determines success/failure and, in case of success, an output value. We denote this value by $P(x, M)$, which is a random variable defined relative to a strategy M for Merlin. Similar to the setting of circuits, we indicate failure by setting $P(x, M) = \bot$, a symbol disjoint from the set of intended output values. Our choice of using success and failure for protocols that output values is to avoid confusion with the decisional notions of acceptance and rejection.

Definition 14 (Arthur-Merlin protocol with output). *Let P be a promise Arthur-Merlin protocol. We say that on a given input $x \in \{0, 1\}^*$:*

- *P outputs v with completeness c if there exists a Merlin strategy such that the probability that P succeeds and outputs v is at least c. In symbols: $(\exists M) \Pr[P(x, M) = v] \geq c$.*
- *P outputs v with soundness s if, no matter what strategy Merlin uses, the probability that P succeeds and outputs a value other than v is at most s. In symbols: $(\forall M) \Pr[P(x, M) \notin \{v, \bot\}] \leq s$.*
- *P has partial single-valuedness s if there exists a value v such that P outputs v with soundness s. In symbols: $(\exists v)(\forall M) \Pr[P(x, M) \notin \{v, \bot\}] \leq s$.*

Note that if P on input x outputs v with completeness c and has partial single-valuedness s, then it outputs v with soundness s, provided $s > 1 - c$. If we omit c and s, then they take their default values of $c = 1$ (perfect completeness) and $s = 1/3$.

For a given function $f : X \to \{0, 1\}^*$ where $X \subseteq \{0, 1\}^*$, we say that P computes f with completeness $c(n)$ and soundness $s(n)$ if on every input $x \in X$, P outputs $f(x)$ with completeness $c(|x|)$ and soundness $s(|x|)$. Note that P may behave arbitrarily on inputs that are not in X. In contrast, an AM protocol (as opposed to a prAM protocol) computing f still computes some value in a complete and sound fashion on inputs $x \notin X$.

3.3 Learn-and-evaluate and commit-and-evaluate protocols

The reconstruction processes for hardness-based hitting-set generators for prAM are typically special types of promise Arthur-Merlin protocols. We distinguish between two types.

A *learn-and-evaluate protocol* is composed of two phases: A *learning* phase followed by an *evaluation* phase. In the learning phase, a probabilistic algorithm makes queries to a function f and produces an output (which we call a sketch). The evaluation phase then consists of a promise Arthur-Merlin protocol that computes $f(x)$ correctly on every input x when given the sketch as additional input.

Definition 15 (Learn-and-evaluate protocol). *A learn-and-evaluate protocol consists of a probabilistic oracle algorithm A_{learn} and a promise Arthur-Merlin protocol P_{eval}. Let $f : X \to \{0,1\}^*$ where $X \subseteq \{0,1\}^*$. We say that (A_{learn}, P_{eval}) computes f with error $e(n)$ for completeness $c(n)$ and soundness $s(n)$ if on every input $x \in X$ of length n the following holds: The probability over the randomness of A_{learn} that P_{eval} with input x and additional input $\pi = A^f_{learn}(1^n)$ outputs $f(x)$ with completeness $c(n)$ and soundness $s(n)$ is at least $1 - e(n)$.*

The learning phase of a learn-and-evaluate protocol can be simulated by an Arthur-Merlin protocol with output, where Merlin guesses the queries that A_{learn} makes on a given random-bit string and answers them in parallel, and the output is a sketch of f. In this view, a learn-and-evaluate protocol becomes a pair of promise Arthur-Merlin protocols: one for the learning phase, and one for the evaluation phase. Note that the quality of the evaluation phase is only guaranteed when the learning queries are answered correctly, i.e., when Merlin is honest in the learning phase.

A *commit-and-evaluate* protocol [SU09] has the syntactic structure of a pair of promise Arthur-Merlin protocols without the restriction that Merlin in the first phase only answers queries about f. Semantically, a commit-and-evaluate protocol is more constrained than a learn-and-evaluate protocol. The first protocol of the pair now represents a commitment phase instead of a learning phase. In this phase, Arthur and Merlin interact and produce an output π, which we call a commitment. Similar to a learn-and-evaluate protocol, the commitment is given as input to the protocol of the evaluation phase. Whereas in a learn-and-evaluate protocol there are no guarantees whatsoever when Merlin is dishonest in the first phase, in a commit-and-evaluate protocol there is a strong guarantee: With high probability over Arthur's randomness in the commitment phase, the evaluation protocol is partial single-valued, meaning that Merlin cannot make Arthur output different values for the same input x with high probability. The guarantee is referred to as resilient partial single-valuedness.

Definition 16 (Commit-and-evaluate protocol). *A commit-and-evaluate protocol is a pair of promise Arthur-Merlin protocols $P = (P_{commit}, P_{eval})$. P has resilience $r(n)$ for partial single-valuedness $s(n)$ on domain $X \subseteq \{0,1\}^*$ if for all n, no matter what strategy Merlin uses during the commit phase, the probability that in the commitment phase, on input 1^n, P_{commit} succeeds and outputs a commitment π that fails to have the following property (3) is at most $r(n)$:*

$$\text{For every } x \text{ of length } n \text{ in } X,\ P_{eval}(x,\pi) \text{ has partial single-valuedness } s(n). \tag{3}$$

In symbols: $(\forall n)(\forall M_{commit})$

$$\Pr[(\forall x \in X \cap \{0,1\}^n) P_{eval}(x,\pi) \text{ has partial single-valuedness } s(n)] \geq 1 - r(n),$$

where $\pi = P_{commit}(1^n, M_{commit})$.

A commit-and-evaluate protocol naturally induces a promise Arthur-Merlin protocol: On input x, run P_{commit} on input $1^{|x|}$. If this process succeeds, let π denote its output and run P_{eval} on input (x, π).

3.4 Bottleneck algorithms

The reconstructor algorithms underlying (targeted) generators typically have the property that they go through a compression phase but eventually produce a potentially long output. We refer to such algorithms as "bottleneck algorithms." We define them generically relative to any base class $\mathcal{A}$ and formalize them as two-phase algorithms: a compression phase A_{comp} that is probabilistic, and a decompression phase A_{dec} that is of type $\mathcal{A}$.

Definition 17. *Let $\mathcal{A}$ be a class of promise algorithms, t a time bound and $s : \mathbb{N} \to \mathbb{N}$. We let $\mathcal{A}\text{TICOMP}[t(n), s(n)]$ be the class of computational problems with the following properties for some probabilistic algorithm A_{comp} and some $A_{dec} \in \mathcal{A}$: For any input $x \in \{0,1\}^*$:*

- *The process first runs A_{comp} on input x, yielding a string $A_{comp}(x)$, and then runs A_{dec} on input $A_{comp}(x)$.*
- *Each of the two phases run in time $t(|x|)$.*
- *The length of $A_{comp}(x)$ never exceeds $s(|x|)$.*

Note that we impose the resource bounds strictly (not up to a constant factor) and on all inputs (not just on all but finitely many). The differences do not matter much for the resource of time. This is because of constant-factor speedup results and because asymptotic time bounds can be turned into absolute ones by hard-wiring the behavior on the finitely many inputs on which the time bound is violated. These transformations do not affect the input-output behavior of the algorithm, though the second one comes at the cost of a potentially significant increase in the description length of the algorithm. For the compression bound $s(n)$ the differences do matter. Constant-factor compression is not possible in general, and hard-wiring is not an option as it requires access to the full input.

Definition 17 applies to promise Arthur-Merlin protocols that output values, yielding the bottleneck protocol classes $\text{prAMTICOMP}[t(n), s(n)]$. In the completeness and soundness notions of Definition 14, for bottleneck protocols, we consider the probabilities over both the internal randomness of the algorithm A_{comp} and Arthur's randomness in the prAM protocol P_{dec}.

We similarly extend the notion of a bottleneck protocol computing a given function f with certain completeness (default 1) and soundness (default 1/3). We say that a pair $(A_{\text{comp}}, P_{\text{dec}})$ is *sound* for a function f if $(A_{\text{comp}}, P_{\text{dec}})$ computes f on every input with soundness 1/3 (without any completeness guarantee).

3.5 Refuters

Refuters and list-refuters can be defined generically for a total function f against a resource-bounded semantic class $\mathcal{A}$ of algorithms. Such $\mathcal{A}$ is defined by an underlying syntactic class of machines, resource bounds that always hold (for all possible executions on all inputs), and promises about the behavior of the machine for it to compute a value on a given input.

Definition 18. *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a total function, and $\mathcal{A}$ a resource-bounded semantic class of algorithms. A list-refuter R for f against $\mathcal{A}$ is an algorithm that on input 1^n and an algorithm A of the syntactic type underlying $\mathcal{A}$, outputs a list of strings $(x_1, \ldots, x_\tau)$, each of length at least n. If A satisfies the resource bounds of $\mathcal{A}$ for all inputs of length at least n, then there exists $i \in [\tau]$ for which $\mathcal{A}$ fails to compute $f(x_i)$. A refuter is a list-refuter that outputs singleton sets.*

Failure for $A(x)$ to compute $f(x)$ means that either A does not satisfy the promise on input x or else it does but computes a value other than $f(x)$.

Other variants on the formal requirements for a refuter exist in the literature; some comments on the choices we made are in order. The lower bound n on the length of the counterexample allows us to avoid irrelevant or useless counterexamples. Such a lower bound could alternately be enforced by modifying A and hard-wiring the correct output values for f on inputs of length less than n. However, this comes at an exponential cost in n for the description length of the algorithm, which is problematic for the efficiency of meta algorithms like refuters. The hard-wiring fix may also not be possible, e.g., in the case of bottleneck algorithms.

Imposing a lower bound rather than an exact value on the length of the counterexamples facilitates handling settings where there are only counterexamples of infinitely many lengths but not all lengths. Note that the length of the counterexamples is bounded by the running time of the refuter, which we typically express as a function of both n and the description length of the algorithm.

In Definition 18 the behavior of a refuter R is well-defined even for algorithms A that do not satisfy the resource constraints on all inputs of length less than n. This is consistent with the requirement that the counterexample be of length at least n. Alternately, one could only specify the behavior of a refuter on algorithms A that satisfy the resource constraints everywhere. For constructible resource bounds, the alternate definition can be used in lieu of ours as one can first modify A into an algorithm A' that satisfies the resource bounds everywhere and behaves like A on inputs where A meets the resource bounds. The increase in description length from A to A' is not significant from a complexity-theoretic perspective. Our definition obviates the need for applying the transformation each time we want to run a refuter.

The refutation problem can have promises beyond the one that A meets the resource bounds on all inputs of length at least n. In such cases the refuter only needs to produce a counterexample when A comes from some restricted subclass of $\mathcal{A}$.

In this work, we mostly use nondeterministic list-refuters against bottleneck Arthur-Merlin protocols, i.e., against classes prAMTICOMP$[t(n), s(n)]$. A nondeterministic list-refuter is similar to a regular list-refuter, with the difference that it is nondeterministic, must have at least one accepting computation path on every input, and must output a list containing a counterexample on every accepting path for every input satisfying the relevant promise. More precisely, on input 1^n and a pair $(A_{\text{comp}}, P_{\text{dec}})$ consisting of a probabilistic algorithm A_{comp} and a prAM protocol P_{dec}, the refuter must have at least one accepting computation path and exhibit the following behavior: every accepting path must output a list $(x_1, \ldots, x_\tau)$, each of length at least n. If on inputs of length $\ell \geq n$ both phases of $(A_{\text{comp}}, P_{\text{dec}})$ run in time $t(\ell)$ and the output length of A_{comp} is bounded by $s(\ell)$, then on every accepting computation path the refuter must output a list of strings $(x_1, \ldots, x_\tau)$, each of length at least n such that for at least one $i \in [\tau]$, $(A_{\text{comp}}, P_{\text{dec}})$ fails to compute f on input x_i with completeness 1 and soundness 1/3.

We say that R is a refuter for f against prAMTICOMP$[t(n), s(n)]$ protocols with *promised soundness* for f if R can refute pairs $(A_{\text{comp}}, P_{\text{dec}})$ that are sound for f. R may fail to refute protocols that are not sound for f, but still needs to have at least one accepting computation path on such inputs.

3.6 Hitting-set generators and targeted hitting-set generators

In the setting of prBPP, Goldreich [Gol20] discusses two equivalent definitions of targeted pseudorandom generators: one for deterministic linear-time machines that take both the input x and the random-bit string r as inputs, and one based on circuits D that only take the random-bit string r as input. The circuit D can be obtained by first constructing a circuit C that simulates the machine on inputs of length $|x|$, and then hardwiring the input x. The difference between a regular and targeted pseudo-random generator lies in the dependency of the output on x (in the first definition) or the circuit D (in the second definition): For a regular PRG the output can only depend on $|x|$ or the size of D, whereas for a targeted PRG it can depend on x and D proper.

In the setting of prAM, without loss of generality, we can assume that promise Arthur-Merlin protocols have perfect completeness. Therefore, we only need to consider targeted hitting-set generators, the variant of targeted PRGs for one-sided error. Similar to the BPP setting, there are two equivalent definitions of targeted HSGs for prAM. We propose a third, hybrid, and also equivalent definition, where the targeted generator is given access to both x and the circuit C. For prAM with perfect completeness the circuit C (as well as D) is co-nondeterministic. For regular HSGs, the output can only depend on the size of C. Our definition highlights that, in principle, there are two types of obliviousness that regular PRGs/HSGs exhibit: With respect to the input (where only dependencies on its size are allowed) and with respect to the algorithm being derandomized (where only dependencies on its running time are allowed). Since the algorithm description can be incorporated as part of the input, the dependency on C can be avoided. This is essentially why all three definitions are equivalent. In our targeted hitting-set generator construction the dependency will only be through x and the size of C.

We start by defining hitting sets for co-nondeterministic circuits.

Definition 19 (Hitting set for co-nondeterministic circuits). *Let D be a co-nondeterministic circuit of size m. A set S of strings of length m is a hitting set for D if there exists at least one $z \in S$ such that $D(z) = 1$ (where D might take a prefix of z as input if necessary). In that case, we say that S hits D.*

The notion allows us to define targeted hitting-set generators for prAM as follows, where we assume, without loss of generality, perfect completeness and soundness 1/2. Regular hitting-set generators are viewed as a special case.

Definition 20 (Regular and targeted hitting-set generator for prAM). *A targeted hitting-set generator for* prAM *is a nondeterministic algorithm that, on input $x \in \{0,1\}^*$ and a co-nondeterministic circuit C, has at least one successful computation path, and if $\Pr_r[C(x,r) = 1] \geq 1/2$, outputs a hitting set for $D(r) \doteq C(x,r)$ on every successful computation path. A regular hitting-set generator for* prAM *is a targeted hitting-set generator where the output only depends on the size of C.*

We measure the running time of a targeted hitting-set generator in terms of both the length n of the string x and the size m of the co-nondeterministic circuit C. In some cases, it is more convenient to work with generators that only take a co-nondeterministic circuit D as input. By the above discussion, such generators suffice for derandomizing prAM.

For completeness, we state the standard way of obtaining the co-nondeterministic circuits C and D capturing promise Arthur-Merlin protocols.

Proposition 21. *There exists an algorithm that, on input 1^n and the description of a (Boolean output, two-round)* prAMTIME$[t(n)]$ *protocol P, runs in time $O(t(n)^2)$ and outputs a co-nondeterministic circuit C of size $m = O(t(n)^2)$ that simulates and negates the computation of P for input length n, i.e., the input of C is comprised of $x \in \{0,1\}^n$ and Arthur's random-bit string r, and it co-nondeterministically verifies that there is no Merlin message that would lead to acceptance. In particular:*

- *If P with input x accepts all random inputs, then $D_x(r) \doteq C(x,r)$ rejects every input.*
- *If P with input x rejects at least a fraction $1/2$ of its random-bit strings, then $D_x(r) \doteq C(x,r)$ accepts at least a fraction $1/2$ of its inputs.*

3.7 PCPs of proximity, error-correcting codes and low-degree extensions

PCPs of proximity work with pair languages, i.e., languages of pairs of strings. Intuitively, we view one part of the input as explicit, to which the PCPP verifier has full access, and another part of the input as implicit, to which the PCPP verifier has oracle access. Each query a PCPP verifier makes to the implicit input counts towards its query complexity.

Let $L \subseteq \{0,1\}^* \times \{0,1\}^*$ be a pair language. We denote by L_x the set $\{z \mid (x,z) \in L\}$. The soundness condition for PCPPs requires that z is sufficiently far from strings in L_x in relative Hamming distance. Let $z, z' \in \{0,1\}^n$ and $d(z,z') = |\{i \mid z_i \neq z'_i\}|/n$. For $z \in \{0,1\}^n$ and $S \subseteq \{0,1\}^n$, we define $d(z,S) = \min_{z' \in S}(d(z,z'))$. The string z is said to be δ-far from S if $d(z,S) \geq \delta$.

Definition 22 (PCP of Proximity). *Let $r,q,t : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and $s, \delta : \mathbb{N} \times \mathbb{N} \to [0,1]$. Let $L \subseteq \{0,1\}^* \times \{0,1\}^*$ be a pair language. We say that $L \in \mathrm{PCPP}_{s,\delta}[r,q,t]$ if there exists a probabilistic algorithm V (the verifier) that, given a string $x \in \{0,1\}^m$ and an integer n as regular input, and oracle access to an implicit input $z \in \{0,1\}^n$ and to a proof oracle $y \in \{0,1\}^*$, tosses $r(m,n)$ coins, queries the oracles z and y for a total of $q(m,n)$ bits, runs in time $t(m,n)$, and either accepts or rejects. Moreover, V has the following properties:*

- *Completeness: If $(x,z) \in L$ then there exists a y such that $\Pr[V^{z,y}(x,n) = 1] = 1$.*
- *Soundness: If (x,z) is such that z is $\delta(m,n)$-far from $L_x \cap \{0,1\}^n$, then for every y' it holds that $\Pr[V^{z,y'}(x,n) = 1] \leq s(m,n)$.*

We use the following PCPP construction due to Ben-Sasson, Goldreich, Harsha, Sudan, and Vadhan.

Lemma 23 ([BSGH+05]). *Let T be a time bound and L be a pair language in $\mathrm{NTIME}[T(m,n)]$, where m denotes the length for the first (explicit) input and n the length for the second (implicit) input. Then, for every constant s, we have $L \in \mathrm{PCPP}_{s,\delta}[r,q,t]$, for*

- *Proximity parameter $\delta(m,n) = 1/\mathrm{polylog}(m,n)$,*
- *Randomness complexity $r(m,n) = \log(1/s) \cdot \log T(m,n) + O(\log\log T(m,n))$,*
- *Query complexity $q(m,n) = \mathrm{polylog}(T(m,n))$,*
- *Proof length $\ell(m,n) = T(m,n) \cdot \mathrm{polylog}(T(m,n))$,*
- *Verification time $t(m,n) = \mathrm{poly}(m, \log n, \log T(m,n))$.*

In our applications of the above PCPP the implicit input will be in an error-correcting format. An *error-correcting code* (ECC) with distance parameter δ is an algorithm Enc such that for every n and $z, z' \in \{0,1\}^n$ for which $z \neq z'$, it holds that $d(\mathrm{Enc}(z), \mathrm{Enc}(z')) \geq \delta$. For our purposes, it suffices that for any constant $\delta \in (0,1]$ there exists an ECC with distance parameter δ that is computable in time $n \cdot \mathrm{polylog}(n)$ (e.g., see [Jus76] and the discussion in [Spi96]).

A particular type of ECCs are *low-degree extensions*. Let $x \in \{0,1\}^n$, $\mathbb{F} = \mathbb{F}_p$ be the field with p elements (for prime p) and h and r integers such that $h^r \geq n$. The low-degree extension of x with respect to p, h, r is the unique r-variate polynomial $\hat{x} : \mathbb{F}^r \to \mathbb{F}$ with degree $h-1$ in each variable, for which $\hat{x}(\vec{v}) = x_i$ for all $\vec{v} \in [h]^r$ representing a $i \in [n]$ and $\hat{x}(\vec{v}) = 0$ for the $\vec{v} \in [h]^r$ that do not represent an index $i \in [n]$. The total degree of $\hat{x}$ is $\Delta = hr$ and $\hat{x}$ is computable in time $n \cdot \mathrm{poly}(h, \log p, r)$ given oracle access to x.

3.8 Average-case simulation

The instance-wise nature of our technique allows us to conclude derandomization on average with respect to arbitrary distributions by assuming hardness with respect to that same distribution. The notion of average-case simulation that we use is the one where the simulation works correctly with high probability over inputs drawn from the distribution. We typically want good simulations to exist with respect to every efficiently sampleable distribution (where the simulation may depend on the distribution). This is usually referred to as the "heuristic" setting.

Definition 24 (Heuristic). *Let Π be a promise-problem, $\mu : \mathbb{N} \to [0,1)$, $\mathcal{C}$ a complexity class and $\mathbf{x} = \{\mathbf{x}_n\}_{n\in\mathbb{N}}$ an ensemble of distributions where $\mathbf{x}_n$ is supported on $\{0,1\}^n$ and such that for all n, every x in the support of $\mathbf{x}_n$ satisfies the promise of Π. We write*

$$\Pi \in \mathrm{Heur}_{\mathbf{x},\mu}\mathcal{C}$$

if there exists a language $L \in \mathcal{C}$ such that for all sufficiently large n, $\Pr_{x\in\mathbf{x}_n}[L(x) \neq \Pi(x)] \leq \mu(n)$. We write

$$\Pi \in \mathrm{Heur}_{\mu}\mathcal{C}$$

if the above property holds for every polynomial-time sampleable ensemble of distributions with the above support restriction.

The notions of average-case simulation extend to the infinitely-often setting in the natural way.

4 Targeted hitting-set generator construction

In this section, we develop our targeted HSG construction, which leads to our instance-wise hardness vs. randomness tradeoffs for Arthur-Merlin protocols.

Our construction builds on the RMV generator due to Shaltiel and Umans [SU09], which is a recursive variant of the MV generator that shares the desired resilience property with MV. We start with the definition of the RMV generator in Section 4.1 and state its reconstruction properties in terms of a commit-and-evaluate protocol. We present our construction and analysis in Section 4.2 and the derandomization consequences in Section 4.3.

4.1 Recursive Miltersen-Vinodchandran generator

We need a couple of ingredients to describe how the RMV generator works. The first one is a local extractor for the Reed-Müller code. A local extractor is a randomness extractor that only needs to know a few bits of the sample. In the following definition the sample is provided as an oracle, and the structured domain from which the sample is drawn is given as an additional parameter.

Definition 25 (Local extractor). *Let S be a set. A (k, ϵ) local S-extractor is an oracle function $E : \{0,1\}^s \to \{0,1\}^t$ that is computable in time $\mathrm{poly}(s,t)$ and has the following property: For every random variable X distributed on S with min-entropy at least k, $E^X(U_s)$ is ϵ-close to uniform.*

We make use of the following local extractor for Reed-Müller codes.

Lemma 26 (Implicit in [SU05]). *Fix parameters $r < \Delta$, and let S be the set of polynomials $\hat{g} : \mathbb{F}^r \to \mathbb{F}$ having total degree at most Δ, where $\mathbb{F} = \mathbb{F}_p$ denotes the field with p elements. There is a $(k, 1/k)$ local S-extractor for $k = \Delta^5$ with seed length $s = O(r \log p)$ and output length $t = \Delta$.*

Note that for every subcube with sides of size $\frac{\Delta}{r}$ and choice of values at its points, there exists an interpolating polynomial $\hat{g}$ with the parameters of Lemma 26. It takes $(\Delta/r)^r \log p$ bits to describe these polynomials, but the local extractor only accesses $\mathrm{poly}(\Delta, r, \log p)$ bits.

When instantiated with a polynomial $\hat{g} : \mathbb{F}^r \to \mathbb{F}$, the RMV generator groups variables and operates over axis-parallel (combinatorial) lines over the grouped variables.[1] Shaltiel and Umans call these MV lines, which we define next.

Definition 27 (MV line). *Let $\mathbb{F} = \mathbb{F}_p$ for a prime p. Given a function $\hat{g} : \mathbb{F}^r \to \mathbb{F}$ where r is an even integer, we define $B = \mathbb{F}^{r/2}$ and identify $\hat{g}$ with a function from B^2 to $\mathbb{F}$. Given a point $\vec{a} = (\vec{a}_1, \vec{a}_2) \in B^2$ and $i \in \{1,2\}$, we define the line passing through $\vec{a}$ in direction i to be the function $L : B \to B^2$ given by $L(\vec{z}) = (\vec{z}, \vec{a}_2)$ if $i = 1$ and $L(\vec{z}) = (\vec{a}_1, \vec{z})$ if $i = 2$. This is an axis-parallel, combinatorial line, and we call it an MV line. Given a function $\hat{g} : \mathbb{F}^r \to \mathbb{F}$ and an MV line L we define the function $\hat{g}_L : B \to \mathbb{F}$ by $\hat{g}_L(z) = \hat{g}(L(z))$.*

The input for the RMV construction is a multivariate polynomial $\hat{g} : \mathbb{F}^r \to \mathbb{F}$ of total degree at most Δ, and the output is a set of m-bit strings for $m \leq \Delta^{1/100}$. The construction is recursive and requires that r is a power of 2 and that p is a prime larger than Δ^{100} (say, between Δ^{100} and $2\Delta^{100}$). Let E be the $(k, 1/k)$-local extractor from Lemma 26 for polynomials of degree Δ in $(r/2)$ variables over $\mathbb{F}$. Remember that $k = \Delta^5$ and that the extractor uses seed length $O(r \log p)$ and output length $t = \Delta \geq m$. By using only a prefix of the output, we have it output exactly m bits.

The operation of the RMV generator on input $\hat{g}$ is as follows: Set $B = \mathbb{F}^{r/2}$. For every $\vec{a} \in B^2$ and $i \in \{1,2\}$, let $L : B \to B^2$ be the MV line passing through $\vec{a}$ in direction i. Compute $E^{\hat{g}_L}(y)$ for all seeds y. For $r = 2$, output the set of all strings of length m obtained over all $\vec{a} \in B^2$, MV lines L through $\vec{a}$, and seeds y. For $r > 2$, output the union of this set and the sets output by the recursive calls $\mathrm{RMV}(\hat{g}_L)$ for each of the aforementioned MV lines L.

The construction runs in time $p^{O(r)}$ and therefore outputs at most that many strings. If the set output by the procedure fails as a hitting set for a co-nondeterministic circuit D of size m, then there exists an efficient commit-and-evaluate protocol for $\hat{g}$ with additional input D. This is the main technical result of [SU09], which we present in a format that is suitable for obtaining our results. Shaltiel and Umans present the evaluation protocol as a multi-round protocol (with $\log r$ rounds). We collapse it into a two-round protocol by standard amplification (which also amplifies the crucial resilience property) [BM88, SU09].

Lemma 28 ([SU09]). *Let Δ, m, r, p be such that $m \leq \Delta^{1/100}$, r is a power of 2 and p is a prime between Δ^{100} and $2\Delta^{100}$. Let also $\mathbb{F} = \mathbb{F}_p$ and $s \in (0,1]$. There exists a commit-and-evaluate protocol (P_{commit}, P_{eval}) with additional inputs p and D, where D is a co-nondeterministic circuit of size m, such that the following holds for any polynomial $\hat{g} : \mathbb{F}^r \to \mathbb{F}$ of total degree at most Δ.*

[1] In the original construction [SU09], the RMV generator is defined with the number d of groups of variables as an additional parameter. Eventually, d is set to 2, which is the value we use for our results as well.

- Completeness: *If D rejects every element output by* $\mathrm{RMV}(\hat{g})$ *then there exists a strategy M_{commit} for Merlin in the commit phase such that P_{eval} on input $(\vec{z}, D, \pi)$ outputs $\hat{g}(\vec{z})$ with completeness 1 for every $\vec{z} \in \mathbb{F}^r$, where $\pi \doteq P_{commit}(1^n, M_{commit})$.*

- Resilience: *If D accepts at least a fraction $1/2$ of its inputs then (P_{commit}, P_{eval}) has resilience s for partial single-valuedness s on domain $\mathbb{F}^r$.*

- Efficiency: *Both P_{commit} and P_{eval} have two rounds. P_{commit} runs in time* $\log(1/s) \cdot \mathrm{poly}(\Delta, r)$ *and P_{eval} runs in time* $(\log(1/s))^2 \cdot \Delta^{O((\log r)^2)}$.

Moreover, the (honest) commitment protocol works as follows: Arthur randomly selects a set $S \subseteq \mathbb{F}^{r/2}$ of size $\log(1/s) \cdot \mathrm{poly}(\Delta, r)$ *and the honest Merlin replies with evaluations of $\hat{g}$ on each of the points in $S^2 \subseteq \mathbb{F}^r$. The honest commitment π consists of the set S and the evaluations of $\hat{g}$ on S^2. Finally, the only way P_{eval} requires access to D is via blackbox access to the deterministic predicate that underlies D.*

4.2 Targeted generator and reconstruction

In this section, we present our targeted HSG construction, which works as follows: On input z and a co-nondeterministic circuit D of size m, it guesses a PCPP (as in Lemma 23) for each bit of $f(z)$ and verifies each PCPP deterministically by enumerating the PCPP verifier's randomness. It encodes the input z with a suitable error-correcting code, obtaining $\mathrm{Enc}(z)$, and instantiates the RMV generator with $\mathrm{Enc}(z)$ and the PCPPs, outputting the union of the outputs for each instantiation. If the generator fails, the reconstruction property for the RMV generator allows for compressing the input z, which is critical for obtaining a bottleneck reconstructor, and the PCPPs, which leads to a more efficient reconstructor. The compressor A_{comp} computes $\mathrm{Enc}(z)$ and the honest commitment π_z for $\mathrm{Enc}(z)$, which is given as input to a prAM protocol P_{dec}, in which Merlin, for any given bit position i, sends a bit b and commits to the low-degree extension of a proof that the i-th bit of $f(z)$ equals b. Arthur then runs the PCPP verifier using the evaluation protocol to answer input and proof queries. The protocol succeeds and outputs b if and only if the PCPP verifier accepts. This approach yields the following statement:

Theorem 29. *Let T be a time bound and f a function computable in nondeterministic time $T(n)$. There exists a nondeterministic algorithm H (the generator) that always has at least one successful computation path per input, and a pair P_{rec} (the reconstructor) consisting of a probabilistic algorithm A_{comp} and a promise Arthur-Merlin protocol P_{dec} such that for every $z \in \{0,1\}^*$ and every co-nondeterministic circuit D that accepts at least half of its inputs, at least one of the following holds.*

1. *$H(z, D)$ outputs a hitting set for D on every successful computation path.*

2. *$P_{dec}(A_{comp}(z, 1^m), D)$ computes $f(z)$ with completeness 1 and soundness 1/3.*

The construction also has the following properties:

- Compression: *On input z of length n and 1^m, the output of A_{comp} has length* $\mathrm{poly}(m, \log T(n))$.

- Resilient soundness: *In both cases 1 and 2 above, the probability that $P_{dec}(D, A_{comp}(1^m, z))$ outputs a value other than $f(z)$ is at most* $1/3$.

- Efficiency: *On input z of length n and 1^m, A_{comp} runs in time* $n \cdot \mathrm{poly}(m, \log T(n))$. *On inputs z of length n and D of size m, H runs in time* $\mathrm{poly}(T(n), m)$ *and P_{dec}, given the output of $A_{comp}(z, 1^m)$ and an additional index i, computes the i-th bit of $f(z)$ in time $(m \cdot \log T(n))^{O((\log r)^2)}$ for $r = O(\log(T(n))/\log m)$. In particular, P_{dec} computes $f(z)$ in time $|f(z)| \cdot (m \cdot \log T(n))^{O((\log r)^2)}$.*

Moreover, $H(z, D)$ only depends on z and the size of D, and the only way P_{dec} requires access to D is via blackbox access to the deterministic predicate that underlies D.

Proof. Fix an input $z \in \{0,1\}^n$. For f computable in nondeterministic time $T(n)$, we define a language L_f that captures the computation of f on inputs encoded with an error-correcting code. Let Enc be an ECC with distance parameter 0.1 computable in time $n \cdot \mathrm{polylog}(n)$ as in Section 3.7. L_f consists of strings $(\tilde{z}, n, i, b)$, where n and i are integers given in binary and $b \in \{0,1\}$, and $\tilde{z} = \mathrm{Enc}(z)$ for $z \in \{0,1\}^n$ such that $f(z)_i = b$. In particular, L_f is decidable in nondeterministic time $n \cdot \mathrm{polylog}(n) + T(n)$ by guessing $z \in \{0,1\}^n$, computing $\mathrm{Enc}(z)$, checking that $\mathrm{Enc}(z) = \tilde{z}$ and computing $f(z)_i$. Let V be the PCPP verifier given by Lemma 23 where we consider $\tilde{z}$ as an implicit input and the remaining part of the input as explicit, with soundness parameter 0.01. The proof length of V is at most $\mathrm{poly}(T(n), n) = \mathrm{poly}(T(n))$ since $T(n) \geq n$. In the following discussion, we let y_i denote any PCPP witnessing $(\tilde{z}, n, i, b) \in L_f$.

We now set parameters for the low-degree extensions that we need. Recall that we wish to instantiate the RMV generator with the low-degree extensions of the PCPPs y_i as well as the encoded input $\tilde{z}$. Given our choice of L_f, the proof length of V is $\mathrm{poly}(T(n))$. To encode the PCPPs, let $h = h(m) = m^{100}$, $r = r(m,n)$ be the smallest power of two such that h^r is greater than or equal to to the proof length of V, and $p = p(m,n)$ the smallest prime in the interval $[\Delta^{100}, 2\Delta^{100}]$ for $\Delta = h \cdot r$, found by exhaustive search. Note, in particular, that $h^r = \mathrm{poly}(T(n), m)$ and $r = O(\log(T(n))/\log m)$. Throughout the rest of the proof, we denote by $\hat{y}_i$ the low-degree extension of each y_i with parameters p, h and r.

To obtain the low-degree extension of $\tilde{z}$, we use slightly different settings. We set h and p as before, but define $r' = r'(m,n)$ to the smallest power of two such that $h^{r'} \geq n$. We denote by $\hat{z}$ the low-degree extension of $\tilde{z}$ with parameters p, h and r'.

Generator. The generator H, on input z and a co-nondeterministic circuit D of size m, computes $\tilde{z} = \mathrm{Enc}(z)$ and the low-degree extension $\hat{z}$ of $\tilde{z}$ with the parameters above, and guesses the value of $v = f(z)$ and a PCPP y_i witnessing $(\tilde{z}, n, i, v_i) \in L_f$ for each index i of v. Then H verifies that $\Pr[V^{\tilde{z}, y_i}(n, i, v_i) = 1] = 1$ for every $i \in [|v|]$ by deterministically enumerating the $\mathrm{poly}(T(n), m)$ random-bit strings for V. If any of the verifications fail, H fails. Otherwise, H computes the low-degree extension $\hat{y}_i$ of y_i. Finally, H outputs the union of $\mathrm{RMV}(\hat{z})$ and $\cup_{i \in [|v|]} \mathrm{RMV}(\hat{y}_i)$, where each invocation of the RMV generator is instantiated with the same output length m. Note that the choice of parameters for encoding $\hat{z}$ and each $\hat{y}_i$ respects the preconditions of Lemma 28.

Computing $\tilde{z}$ and the initial verification step takes time $\mathrm{poly}(T(n), m)$, computing the low-degree extensions for the PCPPs also takes time $\mathrm{poly}(T(n), m)$ and each execution of the RMV generator, including the one for $\hat{z}$, takes time $p^{O(r)} = \mathrm{poly}(T(n), m)$and outputs strings of length m. This culminates in a running time of $\mathrm{poly}(T(n), m)$. Finally, since for the correct output $v = f(z)$ there always exist PCPPs $y_1, \ldots, y_{|v|}$ that are accepted with probability 1 by V, there always exists a nondeterministic guess that leads H to succeed.

Reconstructor. We describe and analyze the pair $(A_{\mathrm{comp}}, P_{\mathrm{dec}})$, which uses the commit-and-evaluate protocol $(P_{\mathrm{commit}}, P_{\mathrm{eval}})$ of Lemma 28 with fixed input p and resilience parameter $s = s(m,n) =$

$(100q)^{-1}$, where $q = q(m,n) = \mathrm{polylog}(T(n),m)$ denotes the query complexity of the PCPP verifier V for L_f on implicit input $\tilde{z}$ and explicit inputs (n,i,b).

On input z and 1^m, A_{comp} first computes $\tilde{z} = \mathrm{Enc}(z)$. Then, A_{comp} tosses the coins required for P_{commit} for the low-degree extension $\hat{z}$ of $\tilde{z}$ and outputs a commitment π_z for $\hat{z}$, which it computes by using the random bits to determine the set S and evaluating $\hat{z}$ on every point of S^2 as in the moreover part of Lemma 28. As for protocol P_{dec}, on input the commitment π_z and an index i, Arthur first tosses the coins required for executing P_{commit} for $\hat{y}_i$. Merlin then replies with a message for P_{commit}, which produces a commitment π_{y_i}, and with a bit b. The honest Merlin should send $b = f(z)_i$ and commit to the low-degree extension of a PCPP y_i that witnesses $f(z)_i = b$, but a dishonest Merlin may send $b \neq f(z)_i$ and/or commit to a different function. Let $\tilde{y}_i$ denote the function that Merlin commits to in the first step, which may be accessed with high probability by executing the evaluation protocol P_{eval} with input π_{y_i}. The restriction of $\tilde{y}_i$ to $[h]^r$ defines a candidate PCPP y_i'. Arthur then runs the PCPP verifier $V^{\tilde{z},y_i'}(n,i,b)$, employing Merlin's help to evaluate $\hat{z}$ and $\tilde{y}_i$ using P_{eval} and the respective commitment whenever V makes a query to $\tilde{z}$ or y_i'. If $V^{\tilde{z},y_i'}(n,i,v_i)$ accepts, then the protocol P_{dec} succeeds and outputs b, otherwise it fails.

Compression. The output of A_{comp} consists of $|S^2| = \log(1/s) \cdot \mathrm{poly}(\Delta, r) = \mathrm{poly}(m, \log T(n))$ points in $\mathbb{F}^{r'}$ together with evaluations of $\hat{z}$ on each point, each of which can be described with $\log p = \mathrm{polylog}(m, \log T(n))$ bits, resulting in a total output length of $\mathrm{poly}(m, \log T(n))$.

Completeness. If D is not hit by $H(z,D)$, then $\mathrm{RMV}(\hat{z})$ fails to hit D and for all indices i there exists at least one proof y_i that witnesses $(\tilde{z},n,i,f(z)_i) \in L_f$ and such that $\mathrm{RMV}(\hat{y}_i)$ fails to hit D. In that case, an honest Merlin can commit to any such $\hat{y}_i$ with probability 1 by the completeness property of Lemma 28. The property also allows the algorithm A_{comp} to compute a correct commitment π_z for $\tilde{z}$ with probability 1. Finally, perfect completeness of V and P_{eval} guarantees that on input π_z and an index i, and when considering an honest Merlin strategy, P_{dec} succeeds and outputs $f(z)_i$ with probability 1.

(Resilient) soundness. If D accepts at least half of its inputs, then the resilience property of the commit-and-evaluate protocol of Lemma 28 guarantees that with probability at least $1-s$, the commitment for $\tilde{y}_i$ is successful, meaning that each execution of the evaluation protocol with input π_{y_i} has partial single-valuedness s. For $\hat{z}$ and the commitment π_z, this implies that with probability at least $1-s$, the evaluation protocol with commitment π_z has soundness s for $\hat{z}$, as A_{comp} always computes the honest commitment. By a union bound, with probability at least $1-2s \geq 0.99$, for sufficiently large m,n, the commit phase is successful for $\hat{z}$ and $\tilde{y}_i$. Let the first "bad" event be the event that at least one of the commitments is unsuccessful. If the first "bad" event does not happen, then by a union bound over the at most q queries made by V to one of $\hat{z}$ or some $\tilde{y}_i$, with probability at least 0.99, every execution of the evaluation protocol results in the evaluation of the respective fixed function. Call the complement of this event the second "bad" event.

Now, the only way Merlin could try to have Arthur output a wrong value, assuming the first two "bad" events do not happen, is if he sends some $b \neq f(z)_i$ in the first round. If this happens, then $(\tilde{z},n,i,b) \notin L_f$, and moreover any $\tilde{w}$ such that $(\tilde{w},n,i,b) \in L_f$ is at relative distance at least 0.1 from $\tilde{z}$. Thus the soundness property of V in Lemma 23 guarantees that P_{dec} fails with probability at least 0.99. Let the third "bad" event be the event that V outputs an incorrect value when the first two "bad" events do not occur. By a union bound over the three "bad" events, all of which have probability at most 0.99, $P_{\mathrm{dec}}(D,(A_{\mathrm{comp}}(1^m,z)))$ either fails or outputs a bit of $f(z)$ with probability at least 2/3. In particular, if completeness also holds then $P_{\mathrm{dec}}(D,(A_{\mathrm{comp}}(1^m,z)))$

computes individual bits of $f(z)$ with completeness 1 and soundness 1/3.

Reconstructor efficiency. The running time for A_{comp} is the time required to compute $\tilde{z} = \text{Enc}(z)$ plus the time required to compute at most $\log(1/s) \cdot \text{poly}(\Delta, r) = \text{poly}(m, \log T(n))$ evaluations of $\hat{z}$. Computing $\tilde{z}$ takes time $n \cdot \text{polylog}(n) = n \cdot \text{polylog}(T(n))$, and computing each evaluation of $\hat{z}$ takes time $n \cdot \text{poly}(h, \log p, r, \log n) = n \cdot \text{poly}(m, \log T(n))$, resulting in a total running time of $n \cdot \text{poly}(m, \log T(n))$.

As for P_{dec}, the commit phase takes time $\log(1/s) \cdot \text{poly}(\Delta, r) = \text{poly}(m, \log T(n))$ and two rounds of communication. Afterwards, evaluating each query made by V with P_{eval} takes time $(\log(1/s))^2 \cdot \Delta^{O((\log r)^2)} = (m \cdot \text{polylog}(T(n)))^{O((\log r)^2)}$. The verification step for V takes time $\text{poly}(m, \log T(n))$, and it makes at most $q = \text{polylog}(m, T(n))$ queries, resulting in a total running time of $(m \cdot \log T(n))^{O((\log r)^2)}$. Moreover, because V's queries are fully determined by its input and random bits, each execution of the evaluation protocol can be carried out in parallel, and thus the total number of rounds is four. Collapsing this protocol into a two-round one using standard techniques [BM88] leads to a prAM protocol with running time $(m \cdot \log T(n))^{O((\log r)^2)}$ with the same completeness and soundness parameters. To compute the entirety of $f(z)$ all at once, we can amplify soundness for P_{dec} by parallel repetition [BM88] so that we still get soundness 1/3 for computing every bit of $f(z)$ in parallel. This introduces a multiplicative overhead of $\text{polylog}(T(n))$ for each execution of P_{dec}, resulting in a total running time of $|f(z)| \cdot (m \cdot \log T(n))^{O((\log r)^2)}$.

Input access. We observe that the only information about D required for computing $\text{RMV}(\hat{z})$ and $\text{RMV}(\hat{y}_i)$ is its size m, and thus the generator H also only requires knowledge of the size of D. Similarly, the commit-and-evaluate protocol in Lemma 28 only requires blackbox access to the deterministic predicate that underlies the circuit D instead of to the description of D, and thus so does P_{dec} since it just passes D as input to the commit-and-evaluate protocol. $\square$

We make some remarks about Theorem 29. First, we could assume that the honest Merlin strategy in protocol P_{dec} knows the value of z. Indeed, if the first "bad" event in the resilient soundness part of the proof of Theorem 29 does not happen, then Merlin is able to reconstruct $\hat{z}$ and thus z from π_z since $\hat{z}$ is the only possible output from running the RMV evaluator with π_z as input. Second, we can amplify the resilient soundness property for the reconstructor via parallel repetition so that the probability that it outputs a value outside of $\{f(z)_i, \perp\}$ is at most 2^{-k}; this incurs a multiplicative running time and compression length overhead of $\Theta(k)$. Finally, we may view the reconstructor as a regular Arthur-Merlin protocol by feeding P_{dec} the input z directly instead of the compressed version π_z. Whenever the PCPP verifier queries $\hat{z}$, Arthur can compute the value by himself in time $\text{poly}(m, n)$, resulting in a final running time of $\text{poly}(n) \cdot (m \cdot \log T(n))^{O((\log r)^2)}$ for computing a bit of $f(z)$ or $|f(z)| \cdot \text{poly}(n) \cdot (m \cdot \log T(n))^{O((\log r)^2)}$ for computing the entirety of $f(z)$.

Stronger resilient soundness. We now present a version of Theorem 29 with a stronger resilient soundness property at the expense of increasing the complexity of the reconstructor from a bottleneck Arthur-Merlin protocol to a probabilistic algorithm with parallel access to SAT. While it is possible to obtain a bottleneck version for this result, it is not necessary since we employ it to obtain our byproducts in the average-case setting, and in this setting having a bottleneck reconstructor does not lead to any stronger results.

Corollary 30. *Let T be a time bound and f a function computable in nondeterministic time $T(n)$. There exists a nondeterministic algorithm H (the generator) that always has at least one successful* *computation path per input, and a probabilistic algorithm A_{rec} (the reconstructor) with parallel access to* SAT *such that for every $z \in \{0,1\}^*$ and every co-nondeterministic circuit D, at least one of the following holds.*

1. *$H(z, D)$ outputs a hitting set for D on every successful computation path.*
2. *$A_{rec}(z, D)$ computes $f(z)$ with probability at least $2/3$.*

The construction also has the following properties:

- Strong resilient soundness: *In both cases 1 and 2 above, the probability that $A_{rec}(z, D)$ outputs a value other than $f(z)$ is at most $1/3$.*
- Efficiency: *On inputs z of length n and D of size m, H runs in time $\mathrm{poly}(T(n), m)$ and A_{rec}, given an additional index i, computes the i-th bit of $f(z)$ in time $\mathrm{poly}(n) \cdot (m \cdot \log T(n))^{O((\log r)^2)}$ for $r = O(\log(T(n))/\log m)$. In particular, A_{rec} computes $f(z)$ in time $|f(z)| \cdot \mathrm{poly}(n) \cdot (m \cdot \log T(n))^{O((\log r)^2)}$.*

Moreover, $H(z, D)$ only depends on z and the size of D, and the only way A_{rec} requires access to D is via blackbox access to the deterministic predicate that underlies D.

The idea behind Corollary 30 is for the reconstructor to first check whether the co-nondeterministic circuit D accepts at least somewhat less than half of its inputs. This is where the parallel access to an oracle for SAT comes in; it allows us to distinguish with high probability between the cases where the fraction of accepted inputs is, say, at most $1/3$ and at least $1/3 + \epsilon$ for some small ϵ. In the former case, the new reconstructor indicates failure with high probability. Otherwise, we boost the fraction of accepted inputs to at least $1/2$ by trying D on two independent inputs, and then run the prAM version of the old reconstructor on the corresponding co-nondeterministic circuit D'.

Proof of Corollary 30. Let H' be the generator and P'_{rec} the regular promise Arthur-Merlin protocol version of the reconstructor of Theorem 29 as discussed in the paragraph after its proof, instantiated with function f and amplified to have (resilient) soundness $1/6$.

Generator. The generator H, on input z and D of size m, first constructs the circuit D' of size $2m$ as $D'(r_1 r_2) = D(r_1) \vee D(r_2)$, where $r_1, r_2 \in \{0,1\}^m$. We then define $H(z, D)$ as $\mathrm{Left}(H'(z, D')) \cup \mathrm{Right}(H'(z, D'))$, where $\mathrm{Left}(S)$ and $\mathrm{Right}(S)$ output the set of the left and right halves of every string in S, respectively.

Reconstructor. On input (z, D) and an index i, the reconstructor A_{rec} estimates up to error $1/12$ and with probability of failure $1/6$ the fraction of inputs accepted by D by evaluating circuit D on $O(1)$ random inputs of length m. This can be done in probabilistic time $\mathrm{poly}(m)$ with $O(1)$ parallel queries to a SAT oracle. If the estimated fraction is less than $5/12$ (the midpoint between $1/3$ and $1/2$), then A_{rec} declares failure. In parallel, A_{rec} builds the circuit D' in the same way as H, selects Arthur's randomness for protocol P'_{rec} with inputs (z, D') and i, and makes three queries to the SAT oracle to obtain the protocol's output: Whether there is a Merlin response that leads to success and whether there are Merlin responses that lead to outputting 0 and 1. If the first query is answered negatively, or the last two queries give inconsistent answers, then A_{rec} declares failure. Otherwise, A_{rec} outputs whatever P'_{rec} does.

Strong resilient soundness. Consider two cases in relation to circuit D: Either D accepts fewer than $1/3$ of its inputs, or it accepts at least $1/3$ of its inputs. In the first case, the initial verification fails with probability at least $5/6$. In the second case, D' accepts at least $1-(2/3)^2 = 5/9 > 1/2$ of its inputs. The resilient soundness property of protocol P'_{rec} guarantees that with probability at least $5/6$, A_{rec} either fails or outputs $f(z)$ correctly. In either case, it follows that A_{rec} outputs an incorrect value for $f(z)$ with probability at most $1/6 < 1/3$.

Correctness. If a co-nondeterministic circuit D accepts at least half of its inputs, so does the circuit D'. Moreover, if $H(z, D)$ fails to hit D, then $H'(z, D')$ fails to hit D'. The correctness of protocol P'_{rec} then guarantees that there exists a strategy for Merlin that makes P'_{rec} output $f(z)$ with probability 1, and no strategy can make P'_{rec} output an incorrect value for $f(z)$ with probability at least $1/6$. In this case, assuming that the fraction of inputs accepted by D was estimated correctly initially, It follows that A_{rec} yields $f(z)$ with probability at least $5/6$. Accounting for the probability of failure of $1/6$ for the estimation, we conclude that A_{rec} outputs $f(z)$ with probability at least $2/3$.

Efficiency. The running time of H is asymptotically identical to that of H', and the running time of A_{rec} is polynomial in the running time of P'_{rec}.

Input access. This part follows right away from the corresponding part of Theorem 29. ☐

Similar to the case of Theorem 29, we can amplify the strong resilient soundness property for the reconstructor of Corollary 30 so that the probability that it outputs a value outside of $\{f(z)_i, \perp\}$ is at most 2^{-k} by running it $\Theta(k)$ times in parallel and outputting the majority answer.

4.3 Derandomization consequences

First, we present a generic derandomization result for prAM that works under hardness against arbitrary distributions.

Theorem 31. *There exists a constant c such that the following holds. Let t, T be time bounds, $\Pi \in \text{prAMTIME}[t(n)]$ and $\{\mathbf{x}_n\}_{n\in\mathbb{N}}$ be an ensemble of distributions such that $\mathbf{x}_n$ is supported over $\{0,1\}^n$ and such that for all n, every x in the support of $\mathbf{x}_n$ satisfies the promise of Π. Assume that for $\mu : \mathbb{N} \to [0, 1)$ there exists a length-preserving function f computable in nondeterministic time $T(n)$ such that for every $\text{prAMTIME}[t(n)^{O((\log r)^2)}]$ protocol P for $r = O(\log(T(n))/\log(t(n)))$, it holds that the probability over $x \sim \mathbf{x}_n$ that $P(x)$ computes $f(x)$ is at most $\mu(n)$ for all but finitely many n. Then, it holds that*

$$\Pi \in \text{Heur}_{\mathbf{x},\mu}\text{NTIME}[T(n)^c].$$

Proof. First, notice that if $t(n) \leq \log T(n)$, then the conclusion is trivial and if $t(n) \geq T(n)$ then the premise is impossible, so we focus on the case that $\log T(n) \leq t(n) \leq T(n)$. Let $\Pi \in \text{prAMTIME}[t(n)]$ and let P_Π be a two-round protocol for Π running in time $O(t(n))$ on inputs of length n. On input $x \in \{0,1\}^n$, compute the circuit D_x of Proposition 21 with protocol P_Π, and note that D_x has size $O(t(n)^2)$. Then, instantiate the HSG of Theorem 29 with f. Feed H inputs x and D_x and run the usual derandomization procedure for protocol P_Π with the set output by $H(x, D_x)$: For each string $\rho \in H(x, D_x)$, nondeterministically guess Merlin's message y_ρ and compute the output of P_Π with randomness ρ and message y_ρ, accepting if and only if P_Π accepts for every

$\rho \in H(x, D_x)$. The entire procedure runs in nondeterministic time $\mathrm{poly}(T(n), t(n)) = O(T(n)^c)$ for some constant c, since $T(n) \geq t(n)$.

Assume, with the intent of deriving a contradiction, that with probability at least $\mu(n)$ over $x \sim \mathbf{x}_n$, this derandomization fails for input x. First, notice that by the perfect completeness of P_Π it must be the case that such an x lies in Π_N and that P_Π with input x accepts every string in $H(x, D_x)$. Therefore, D_x acts as a distinguisher for $H(x, D_x)$, i.e., it rejects every string output by D_x while accepting at least half of its inputs. By computing D_x and feeding it to the regular prAM protocol version P_{rec} of the reconstructor of Theorem 29, we obtain a prAM protocol that computes individual bits of $f(x)$ correctly for every x for which the derandomization fails, i.e., with probability at least $\mu(n)$ over $x \sim \mathbf{x}_n$. By running this protocol n times in parallel to compute every bit of $f(x)$, we obtain a prAM protocol that runs in time

$$\mathrm{poly}(n) \cdot (t(n) \cdot \log T(n))^{O((\log r)^2)} = t(n)^{O((\log r)^2)}$$

since $t(n) \geq \log T(n)$ and $t(n) \geq n$. This is a contradiction to the hardness of f so we are done. □

We remark that we require hardness not just against AM protocols but against prAM protocols, which may not respect the completeness and/or soundness conditions on some inputs. However, an input of length n only contributes to the success fraction $\mu(n)$ provided the completeness and soundness conditions are met on that input.

As a consequence of Theorem 31, if the hardness assumption holds for almost-all inputs, then we obtain full derandomization of prAM.

Theorem 32. *There exists a constant c such that the following holds. Let t, T be time bounds. If there is a length-preserving function f computable in nondeterministic time $T(n)$ that is hard on almost-all inputs against* $\mathrm{prAMTIME}[t(n)^{O((\log r)^2)}]$ *for $r = O(\log(T(n))/\log(t(n)))$ then there exists a targeted hitting-set generator that achieves the derandomization*

$$\mathrm{prAMTIME}[t(n)] \subseteq \mathrm{NTIME}[T(n)^c].$$

Proof. The statement follows from Theorem 31 by noting that the assumption that f is hard on almost-all inputs implies that, for sufficiently large n, f is hard for all possible distributions $\mathbf{x}_n$ with success probability $\mu(n) = 0$. In particular, the following nondeterministic algorithm is a hitting-set generator for prAM: On input $x \in \{0,1\}^n$ and a co-nondeterministic circuit C of size m, output $H(x, D)$ where H is the generator of Theorem 29 and $D \doteq C(x, \cdot)$. This algorithm has a successful computation path for any input and, on every successful computation path on inputs where D accepts at least half of its inputs, it outputs a set that hits D. The running time of the generator is $\mathrm{poly}(T(n), m)$, which results in the derandomization result in the theorem statement when $T(n) \geq m$. □

By setting parameters in Theorem 32, we obtain the derandomization results listed on Table 2. In particular, the first line of Table 2 establishes Theorem 6 and the last line establishes Theorem 7. We now provide more details on how to obtain each line of Table 2:

- For the high end, set $t(n) = n$, in which case $r = O(a)$. Then, $\mathrm{prAMTIME}[n] \subseteq \mathrm{NP}$ follows as long as f is hard on almost-all inputs against $\mathrm{prAMTIME}[n^{O((\log a)^2)}]$. The result for prAM follows by padding.

Setting	$T(n)$	Hard for	Derandomization
high end	n^a	$n^{O((\log a)^2)}$	$\mathrm{prAM} \subseteq \mathrm{NP}$
middle-of-the-road	$2^{\mathrm{polylog}(n)}$	$n^{O((\log\log n)^2)}$	$\mathrm{prAM} \subseteq \mathrm{NTIME}[2^{\mathrm{polylog}(n)}]$
low end	$2^{n^{o(1)}}$	$n^{o((\log n)^2)}$	$\mathrm{prAM} \subseteq \mathrm{NTIME}[2^{n^{o(1)}}]$
very low end	$2^{\mathrm{poly}(n)}$	$n^{b(\log n)^2}\ \forall b$	$\exists c\ \mathrm{prAM} \subseteq \mathrm{NTIME}[2^{n^c}]$

Table 2: Derandomization consequences that follow from different instantiations of Theorem 32.

- For the middle-of-the-road result, set $t(n) = n$, in which case $r = \mathrm{polylog}(n)$. Then, $\mathrm{prAMTIME}[n] \subseteq \mathrm{NTIME}[2^{\mathrm{polylog}(n)}]$ follows as long as f is hard on almost-all inputs against $\mathrm{prAMTIME}[n^{O((\log\log n)^2)}]$. The result for prAM follows by padding.
- For the low end, let $\nu = \nu(n) = o(1)$ be such that $T(n) = 2^{n^\nu}$ and set $t(n) = n$. In this case, $r \leq n^\nu$. Then, $\mathrm{prAMTIME}[n] \subseteq \mathrm{NTIME}[\mathrm{poly}(n, 2^{n^\nu})]$ follows as long as f is hard on almost-all inputs against $\mathrm{prAMTIME}[n^{O((\nu\log n)^2)}]$. Since $\mathrm{poly}(n, 2^{n^\nu}) = 2^{n^{o(1)}}$ and $n^{O((\nu\log n)^2)} = n^{o((\log n)^2)}$, the result for prAM follows by padding.
- For the very low end, set $t(n) = n^b$ for a constant b, in which case $r = \mathrm{poly}(n)$. Then, $\mathrm{prAMTIME}[n^b] \subseteq \mathrm{NTIME}[2^{n^c}]$ for some constant c follows as long as f is hard on almost-all inputs against $\mathrm{prAMTIME}[n^{O(b(\log n)^2)}]$. To get the result for prAM, it suffices for hardness to hold for all constants b.

4.4 From refutation to derandomization

In this section, we show that the second item in Theorem 8 implies the third one.

Here is the outline for the construction of the targeted hitting-set generator for prAM, assuming a refuter for a function f computable in nondeterministic time n^a. On input a co-nondeterministic circuit D of size m, we first run the assumed list-refuter on the input consisting of 1^n for a sufficiently large n and the reconstructor protocol P_{rec} from Theorem 29 with D fixed. This produces a list of strings $z_1, \ldots, z_\tau$, each of length at least n. We use each of them as an input for the generator H of Theorem 29 and output the union of the sets obtained. Provided that n is a sufficiently large polynomial in m, the reconstructor meets the resource bounds for a $\mathrm{prAMTICOMP}[n^{a+\epsilon}, n^\epsilon]$ protocol at length n. The defining property of the list-refuter then guarantees that for at least one z_i, the reconstructor fails to compute $f(z_i)$ (item 2 in Theorem 29 fails for z_i). It follows that $H(z_i, D)$ hits D (item 1 in Theorem 29 holds).

Theorem 33. *Let T be a time bound, a a constant and f a function computable in nondeterministic time n^a. If for some constant $\epsilon \in (0,1)$ there is a nondeterministic list-refuter R for f against* $\mathrm{prAMTICOMP}[n^{a+\epsilon}, n^\epsilon]$ *protocols with promised soundness for f such that R runs in time T, then there is a targeted hitting-set generator for* prAM *that is computable in nondeterministic time* $\mathrm{poly}(T(\mathrm{poly}(n)))$.

Proof. Let $(A_{\mathrm{comp}}, P_{\mathrm{dec}})$ be the reconstructor of Theorem 29 instantiated with f. We first describe the operation of the targeted HSG, then we analyze its correctness and running time.

Generator. The generator, on input a co-nondeterministic circuit D of size m, first sets $n = n(m)$ to be determined later. Let $A_{\mathrm{comp}}(\cdot, 1^m)$ denote algorithm A_{comp} with 1^m fixed as its second input and similarly let $P_{\text{dec}}(\cdot, D)$ be the protocol P_{dec} with the circuit D fixed as its second input. The generator then feeds inputs 1^n and $(A_{\text{comp}}(\cdot, 1^m), P_{\text{dec}}(\cdot, D))$ into the refuter R to obtain a list of inputs $(z_1, \ldots, z_\tau)$. Finally, the generator outputs $\cup_{i\in[\tau]} H(z_i, D)$, where H is the generator of Theorem 29 instantiated with f. Observe that the generator always has a successful computation path for every input since so does the refuter R.

Correctness. Note that as long as D accepts at least half of its inputs, the resilient soundness property in Theorem 29 guarantees that $(A_{\text{comp}}(\cdot, 1^m), P_{\text{dec}}(\cdot, D))$ is sound for f. To ensure correctness of the generator, we set the value of n sufficiently large such that $A_{\text{comp}}(\cdot, 1^m)$ and $P_{\text{dec}}(\cdot, D)$ run in time at most $n^{a+\epsilon}$ and such that the output length of $A_{\text{comp}}(\cdot, 1^m)$ is at most n^ϵ. In this case, the refuter must output, on every accepting computation path, a list of strings $(z_1, \ldots, z_\tau)$ that contains at least one z_i such that $(A_{\text{comp}}(\cdot, 1^m), P_{\text{dec}}(\cdot, D))$ fails to compute $f(z_i)$ with completeness 1 and soundness $1/3$. This means that item 2 in Theorem 29 fails for $z = z_i$, and therefore item 1 must hold, which implies that our targeted generator hits D.

We now set the value of n. We set $n = m^k$, where k is a constant that respects the lower bounds we set in the following discussion. Recall that, on input $z \in \{0,1\}^n$, $A_{\text{comp}}(\cdot, 1^m)$ outputs a string of length $\text{poly}(m, a \log n) \le (m \cdot \log n)^{k_1}$ for a fixed constant k_1. Moreover, the running time for $A_{\text{comp}}(\cdot, 1^m)$ is $n \cdot \text{poly}(m, a \log n) = n \cdot \text{poly}(m, \log n) \le n \cdot (m \cdot \log n)^{k_2}$ for some constant k_2, and the running time for $P_{\text{dec}}(\cdot, D)$ is $n^a \cdot (m \cdot a \log n)^{O((\log r)^2)}$ for $r = O(a \log n / \log m)$, and thus upper bounded by $n^a \cdot (m \cdot \log n)^{k_3 \cdot (\log(a \log n/\log m))^2}$ for some constant k_3.

By setting $k \ge 2k_1/\epsilon$, it holds for sufficiently large m and any input of length $\ell \ge n = m^k$ that the string output by $A_{\text{comp}}(\cdot, 1^m)$ has length at most ℓ^ϵ. Similarly, setting $k \ge 2k_2/\epsilon$, it holds for for sufficiently large m and any input of length $\ell \ge n = m^k$ that the running time of $A_{\text{comp}}(\cdot, 1^m)$ is at most $\ell^{1+\epsilon} \le \ell^{a+\epsilon}$. Finally, setting $k \ge 2k_3 \cdot (\log(ak))^2/\epsilon$, which holds for sufficiently large constant k, guarantees that $P_{\text{dec}}(\cdot, D)$ runs in time at most $\ell^{a+\epsilon}$ for $\ell \ge n = m^k$ and sufficiently large m.

Running time. Let the constant c denote the description length of $(A_{\text{comp}}, P_{\text{dec}})$, it follows that $(A_{\text{comp}}(\cdot, 1^m), P_{\text{dec}}(\cdot, D))$ has description length at most $m' = c + \Theta(m \log m) = \Theta(m \log m)$. Computing the list of inputs $(z_1, \ldots, z_\tau)$ using the refuter R takes time $T(n + m') = T(\text{poly}(m))$, which also serves as an upper bound for the length of each z_i. Finally, computing $H(z_i, D)$ for all z_i takes time $\text{poly}(T(\text{poly}(m)))$, which dominates the running time for the generator. $\square$

Theorem 33 scales very smoothly with respect to the time T for computing the refuter. In particular, it allows us to obtain equivalences at the low end of the derandomization spectrum. Apart from the time T for computing the refuter, one can also vary the time for computing f as well as the compression length for the bottleneck protocols. Increasing the time required for computing f leads to a similar increase in the time bound for the class against which we require refuters. Decreasing the compression length requires the targeted HSG to run the refuter with a larger input length n. Due to the sub-optimal scaling of the RMV reconstructor, our approach does not work for equivalences at the low end in either direction. It does work for intermediate ranges, e.g., for running time bounds of the form $2^{\text{polylog}(n)}$ and compression lengths of the form $2^{(\log n)^\epsilon}$ for $\epsilon \in (0,1)$.

Theorem 34. *Let a be a constant and f a function computable in nondeterministic time $2^{(\log n)^a}$. If for some constant $\epsilon \in (0,1)$ there is a nondeterministic list-refuter R for f against protocols in* $\text{prAMTICOMP}[2^{(\log n)^{a+\epsilon}}, 2^{(\log n)^\epsilon}]$ *with promised soundness for f such that R runs in time*

$2^{\mathrm{polylog}(n)}$*, then there is a targeted hitting-set generator for* prAM *that is computable in nondeterministic time* $2^{\mathrm{polylog}(n)}$.

5 Consequences of derandomization

In this section, we prove the directions of derandomization to hardness and derandomization to targeted HSGs of our near-equivalences, as well as the direction of targeted HSGs to refutation of our equivalence.

5.1 Hardness on almost-all inputs

We start with our derandomization-to-hardness implication: If prAM $\subseteq$ NP then for all constants c there is a length-preserving function f computable in nondeterministic polynomial time (with a few bits of advice) that is hard on almost-all inputs against AMTIME$[n^c]$. The basic idea is that, under the derandomization hypothesis, every (single-bit) AM protocol that runs in time n^c can be simulated by a single-valued nondeterministic machine without too much time overhead. If we have as advice whether a particular nondeterministic machine is single-valued or not at input length n, we can negate its input efficiently, obtaining a function f computable in nondeterministic time poly(n) that is almost-all inputs hard against AM protocols that run in time n^c. We now state Proposition 5 formally.

Proposition 35 (Formal version of Proposition 5). *If* prAM $\subseteq$ NP*, then for every constant c and increasing function $\alpha : \mathbb{N} \to \mathbb{N}$ there exists a length-preserving function f computable in nondeterministic polynomial time with $\alpha(n)$ bits of advice that is hard on almost-all inputs against* AMTIME$[n^c]$.

Proof. Assume that prAM $\subseteq$ NP and let c' be a constant to be defined later (which depends on c). The basic idea for the function f is as follows: On an input z of length n, we set its i-th output bit (for $1 \le i \le \min(n, \alpha(n))$) to the opposite of the i-th bit output by the i-th nondeterministic Turing machine N_i on input z (if N_i is single-valued and halts in at most $n^{c'+2}$ steps at input length n), and otherwise we set it to 0. Formally, on input z of length n and for $1 \le i \le n$

$$f(z)_i = \begin{cases} 1 - N_i(z)_i & \text{if } i \le \alpha(n),\ N_i \text{ is single-valued and halts in at most } n^{c'+2} \text{ steps,} \\ 0 & \text{otherwise.} \end{cases}$$

Note that f is computable by a single-valued nondeterministic machine running in time $O(n^{c'+3})$ with $\alpha(n)$ bits of advice (indicating whether N_i is single-valued and halts in at most $n^{c'+2}$ steps at input length n for $1 \le i \le \alpha(n)$). The nondeterministic machine computing f is only guaranteed to be single-valued when given the correct advice string. This holds because, when N_i is single-valued, computing $1 - N_i(z)_i$ can be done by guessing a path on which N_i succeeds, which must result in the unique value $N_i(z)$, and then outputting the opposite of the i-th bit of that. Assume, with the intent of deriving a contradiction, that there exists an AM protocol P that runs in time $O(n^c)$ and computes f on an infinite set of inputs $Z \subseteq \{0,1\}^*$. Consider the protocol P' that takes as regular input a triple (z, i, b) and accepts iff the i-th bit of the output of protocol P with input z equals b (if $i > |z|$ then P' rejects). Note that P' induces a language L in AMTIME$[n^c]$. Since prAM $\subseteq$ NP and prAMTIME$[n^c]$ has a complete problem under linear-time reductions, it follows that there exists a constant c' such that AMTIME$[n^c] \subseteq$ NTIME$[n^{c'}]$. While our argument only requires that there exists a constant c' such that AMTIME$[n^c] \subseteq$ NTIME$[n^{c'}]$, we use the assumption prAM $\subseteq$ NP instead of AM $\subseteq$ NP since it is unknown whether AMTIME$[n^c]$ contains a complete problem under linear-time reductions.

Let N be a nondeterministic machine that runs in time $n^{c'}$ and computes L. Note that for every $z \in \{0,1\}^*$ and $1 \leq i \leq |z|$, $N(z,i,b) = 1$ for exactly one $b \in \{0,1\}$, and when $z \in Z$, $N(z,i,b) = 1$ if and only if $f(z)_i = b$. Now consider the following procedure N': On input $z \in \{0,1\}^n$, guess a value b_i and a witness y_i for each $1 \leq i \leq n$ and run $N(z,i,b_i;y_i)$. If for all i, $N(z,i,b_i;y_i)$ accepts, N' succeeds and prints the concatenation of the guessed b_i's, otherwise N' fails. Note that N' is a nondeterministic machine that runs in time $O(n^{c'+1})$. Moreover, by our assumption that P is an AM protocol and that prAM $\subseteq$ NP, N' is single-valued on *every* input. By construction, the single value equals $f(z)$ for all $z \in Z$.

Let i be the index of N' in our enumeration, i.e., $N_i = N'$. By definition of f, for every input $z \in \{0,1\}^*$ of sufficiently large length $n \geq \alpha^{-1}(i)$ (so that it has a chance to negate the output of N_i), and in particular for all sufficiently large $z \in Z$, we have that $f(z)_i = 1 - N'(z)_i = 1 - f(z)_i$, a contradiction. □

This result extends to other parameter settings. As an example, we state a version of Proposition 35 at the very low end.

Proposition 36. *If there exists a constant c such that that* AM $\subseteq$ NTIME$[2^{n^c}]$*, then for every increasing function $\alpha : \mathbb{N} \to \mathbb{N}$ there exists a function f computable in nondeterministic exponential time with $\alpha(n)$ bits of advice that is hard on almost-all inputs against* AM *protocols running in polynomial time.*

Proof (Sketch). The proof is essentially identical to that of Proposition 35, but with a different time bound. Since AM $\subseteq$ NTIME$[2^{n^c}]$, the diagonalizing machine N needs to diagonalize against single-valued nondeterministic algorithms running in time $2^{n^{c'}}$ for some fixed constant $c' > c$, and thus we get a nondeterministic algorithm that runs in time $O(2^{n^k})$ for any constant $k > c'$. □

We conclude this section by pointing out the remaining gaps between the direction from hardness to derandomization and the reverse direction in the setting of hardness on almost-all inputs. The first gap lies in the fact that in the derandomization-to-hardness direction, the function f requires a few bits of advice that we don't know how to handle in the other direction. A subtler gap relates to the difference between AM and prAM. In the hardness-to-derandomization direction, we require hardness against prAM protocols, which may not obey the AM promise on all inputs. In the derandomization-to-hardness direction, we can only guarantee hardness against AM protocols, which necessarily obey the AM promise on all inputs. We remark that a similar problem shows up in other hardness vs. randomness tradeoffs for AM [GSTS03, SU09]. For example, to conclude almost-everywhere derandomization of AM, the authors of [GSTS03] require hardness of EXP against AM protocols for which completeness only holds infinitely-often. Finally, we also note that, while Chen and Tell only state their derandomization-to-hardness result for BPP [CT21], in that setting one can actually achieve hardness against prBPP (where the probabilistic algorithm might not have a high-probability output for every input).

5.2 Targeted hitting-set generator

In this section, we prove Theorem 9 along the lines of the intuition provided in Section 2.2. We make use of a win-win argument: Either the EXP $\neq$ NEXP hardness assumption holds, in which case there is a regular (oblivious) HSG that guarantees the derandomization result [IKW02]. Or else we may assume that EXP = NEXP, which allows us to construct a function f that is hard against prAM protocols by diagonalization, with which we then instantiate Theorem 29 to obtain the targeted HSG.

We need the following result that follows from the "easy-witness" method.

Lemma 37 ([IKW02]). *If* NEXP $\neq$ EXP *then* prAM $\subseteq$ io-NTIME$[2^{n^\epsilon}]/n^\epsilon$ *for every* $\epsilon > 0$. *Moreover, there exists a (regular) HSG that achieves this derandomization.*

We now prove Theorem 9, which we restate here for convenience.

Theorem 9 (Restated). *If* prAMTIME$[2^{\mathrm{polylog}(n)}] \subseteq$ io-NEXP*, then there exists a targeted hitting-set generator that yields the simulation* prAM $\subseteq$ io-NTIME$[2^{n^c}]/n^\epsilon$ *for some constant* c *and all* $\epsilon > 0$.

Proof. If EXP $\neq$ NEXP, we are done by Lemma 37. Otherwise, it holds that NEXP = EXP. We use this collapse to construct a length-preserving multi-bit function f computable in exponential time that is hard against prAMTIME$[n^{(\log n)^3}]$. We then instantiate Theorem 31 with f to obtain the targeted HSG. Hardness against protocols running in this time bound suffices along the lines of Theorem 7.

Before constructing f, we make an observation: Due to the instance-wise nature of our construction, to obtain an infinitely-often derandomization result using Theorem 31 it suffices to have an *infinitely-often all-inputs hardness assumption.* More precisely, we require the following: For every prAMTIME$[n^{(\log n)^3}]$ protocol P, there exist infinitely many input lengths n such that P fails to compute f for every z of length n. Thus, we construct a function f with this requirement in mind.

Under the hypothesized derandomization assumption and because prAMTIME$[n^{(\log n)^3}]$ has a complete problem under linear-time reductions, it follows that there exists a constant k such that prAMTIME$[n^{(\log n)^3}] \subseteq$ io-NTIME$[2^{n^k}]$. Since NTIME$[2^{n^k}]$ also has a complete problem under linear-time reductions, under the assumption EXP = NEXP, there exists a constant k' such that prAMTIME$[n^{(\log n)^3}] \subseteq$ io-DTIME$[2^{n^{k'}}]$. In that case, it suffices to diagonalize against fixed-exponential time machines to construct f. Similar to Proposition 35, we define the i-th bit of $f(z)$ to be the opposite of the i-th bit output by $M_i(z)$ when it runs for at most $2^{|z|^{k'+1}}$ steps, where M_i is the i-th deterministic Turing machine. Formally, on input z of length n and for $1 \leq i \leq n$,

$$f(z)_i = \begin{cases} 1 - M_i(z) & \text{if } M_i(z) \text{ halts in at most } 2^{n^{k'+1}} \text{ steps,} \\ 0 & \text{otherwise.} \end{cases}$$

Note that f is computable by a deterministic machine running in time $O(n \cdot 2^{n^{k'+1}})$.

Assume, with the intent of deriving a contradiction, that there exists a prAMTIME$[n^{(\log n)^3}]$ protocol P such that for almost-all input lengths n, P computes f on at least one input $z \in \{0,1\}^n$, and call the set of inputs where P computes f correctly Z. Again, similar to the proof of Proposition 35, P induces a problem Π in prAMTIME$[n^{(\log n)^3}]$, and by our assumptions, there is a language $L \in \mathrm{DTIME}[2^{n^{k'}}]$ such that L and Π agree on infinitely many input lengths. Let M be a deterministic Turing machine running in time $O(2^{n^{k'}})$ that decides L. Recall that yes-instances of Π are triples (z, i, b) such that $z \in Z$ and $f(z)_i = b$ while no-instances have $z \in Z$ and $f(z)_i \neq b$. Let M' be the deterministic Turing machine that, on input z of length n, outputs $M'(z)$ of length n such that $M'(z)_i = 1$ if and only if M accepts $(z, i, 1)$ for $1 \leq i \leq n$. Note that M' runs in time $2^{n^{k'+1}}$. By construction and our assumption on P, for infinitely many input lengths n there exists at least one $z \in Z \cap \{0,1\}^n$ such that $M'(z) = f(z)$. Let i be the index of M' in our enumeration. By definition of f, for every input $z \in \{0,1\}^*$ of sufficiently large length $n \geq i$ (so that it has a chance to negate the output of M'), and in particular for all sufficiently large inputs $z \in Z$, we have that $f(z)_i = 1 - M'(z)_i = 1 - f(z)_i$, which gives us the sought contradiction.

Under the hypotheses of the theorem, we have constructed a length-preserving function f that is computable in exponential time T with the property that for every $\mathrm{prAMTIME}[n^{(\log n)^3}]$ protocol P there are infinitely many lengths n such that on every input $z \in \{0,1\}^n$, P fails to compute f. We instantiate Theorem 31 with f to obtain a targeted HSG for prAM that, on input $z \in \{0,1\}^n$ and a co-nondeterministic circuit D of size m, runs in time $\mathrm{poly}(T(n), m)$ and works for all inputs z of infinitely-many input lengths n, resulting in a fixed-exponential time nondeterministic simulation for prAM that works for infinitely many input lengths. □

5.3 Refuters from targeted hitting-set generators

In this section, we prove that the third item in Theorem 8 implies the first one. In fact, we establish something stronger: Assuming the existence of a targeted hitting-set generator as in the third item, every function f that is computable in nondeterministic polynomial-time and has a *probabilistic* polynomial-time refuter against bottleneck protocols with imperfect completeness and promised soundness for f, also has a nondeterministic polynomial-time list-refuter against the same class but with the standard perfect completeness level (Theorem 38). The first item then follows as the identity function has such a probabilistic refuter (Proposition 39).

A probabilistic refuter is a refuter that produces a counterexample with constant probability over its internal randomness. In the case of the class $\mathrm{prAMTICOMP}[t(n), s(n)]$ with imperfect completeness level $c = 2/3$ (and default soundness level $s = 1/3$), this means the following: On input 1^n and a pair $(A_{\mathrm{comp}}, P_{\mathrm{dec}})$ consisting of a probabilistic algorithm A_{comp} and a prAM protocol P_{dec}, a probabilistic refuter for a function f outputs a string z of length at least n such that the following holds with probability $\Omega(1)$. If on inputs of length $\ell \geq n$ both phases of $(A_{\mathrm{comp}}, P_{\mathrm{dec}})$ run in time $t(\ell)$ and the output length of A_{comp} is bounded by $s(\ell)$, then $(A_{\mathrm{comp}}, P_{\mathrm{dec}})$ fails to compute f on input z with completeness $2/3$ and soundness $1/3$. Note that if $(A_{\mathrm{comp}}, P_{\mathrm{dec}})$ is promised to be sound for f and to obey the time and compression requirements, then it must be the case that $(A_{\mathrm{comp}}, P_{\mathrm{dec}})$ fails to compute $f(z)$ with completeness $2/3$.

Here is the intuition for the stronger statement (Theorem 38). To derandomize the given probabilistic refuter, we set up a co-nondeterministic circuit D that verifies that a random bit-string leads to a counterexample for a given bottleneck Arthur-Merlin protocol $(A_{\mathrm{comp}}, P_{\mathrm{dec}})$ with promised soundness for f. On input a string (r_1, r_2, r_3) where r_1 represents the randomness for the probabilistic refuter R_{pr}, r_2 the randomness for A_{comp} and r_3 the randomness for P_{dec}, D first computes the candidate counterexample z by running R_{pr} and uses co-nondeterminism to determine $f(z)$. D then co-nondeterministically verifies that all possible replies from Merlin would lead P_{dec} with input $A_{\mathrm{comp}}(z, r_2)$ and randomness r_3 to fail or output something other than $f(z)$.

If $(A_{\text{comp}}, P_{\text{dec}})$ is sound for f and obeys the time and compression requirements, the only way the refuter can succeed is when $(A_{\text{comp}}, P_{\text{dec}})$ fails the completeness requirement on z. Since the refuter succeeds with probability $\Omega(1)$ and the completeness level is bounded below 1, this means that the circuit D accepts a $\Omega(1)$ fraction of its inputs. Thus, when we apply the assumed targeted hitting-set generator to D, it has to output at least one (r_1, r_2, r_3) on which D succeeds. For such a (r_1, r_2, r_3), $(A_{\text{comp}}, P_{\text{dec}})$ does not have perfect completeness on the input z that R_{pr} produces with random-bit string r_1 because $(A_{\text{comp}}, P_{\text{dec}})$ does not output $f(z)$ on random bit-string (r_2, r_3). Thus, outputting the strings z over all (r_1, r_2, r_3) that the targeted HSG produces, yields the desired nondeterministic polynomial-time list-refuter.

Note the increase in the completeness level from $c = 2/3$ for a probabilistic refuter to $c = 1$ in the corresponding item for a nondeterministic refuter as in Section 3.5. On the one hand, the gap in completeness for the counterexample output by a probabilistic refuter allows the co-nondeterministic circuit D to accept a constant fraction of its inputs, which is needed to guarantee success for the derandomization. On the other hand, the nondeterministic refuter we obtain from the probabilistic refuter only guarantees that the completeness on the counterexample is not perfect. The latter guarantee suffices for the direction from refutation to derandomization because the reconstructor in Theorem 29 has perfect completeness. The resilient soundness property of the reconstructor in Theorem 29 ensures that we only need to worry about refuting pairs $(A_{\text{comp}}, P_{\text{dec}})$ that are sound for f.

Theorem 38. *Assume that there is a targeted hitting-set generator for* prAM *computable in nondeterministic time T. Let f be a function computable in nondeterministic polynomial time that has a probabilistic polynomial-time refuter against* prAMTICOMP$[t(n), s(n)]$ *protocols with promised soundness for f. There exists a list-refuter R for f against* prAMTICOMP$[t(n), s(n)]$ *protocols with promised soundness for f such that R is computable in nondeterministic time* $T(\text{poly}(m, t(\text{poly}(n))))$, *where m denotes the description length of the protocol to be refuted and n the lower bound for the length of the counterexamples.*

We observe that in case the function f in Theorem 38 is computable in polynomial time, as is the case with identity, the list-refuter R runs in polynomial time in its input length, i.e., in time $\text{poly}(m, n)$.

Proof of Theorem 38. Let H be the hypothesized targeted HSG. H always has an accepting computation path for any input, and on input a co-nondeterministic circuit D of size m' that accepts at least $1/2$ of its inputs, it runs in time $T(m')$ and outputs, on every accepting computation path, a set S that hits D. Let R_{pr} be the hypothesized probabilistic refuter for f against prAMTICOMP$[t(n), s(n)]$ protocols with promised soundness for f, and assume w.l.o.g. that R_{pr} only outputs strings of length at least n and succeeds in outputting a counterexample with constant probability $\delta > 0$.

We now describe the nondeterministic list-refuter R. The input is 1^n and a pair $(A_{\text{comp}}, P_{\text{dec}})$ consisting of a probabilistic algorithm A_{comp} and a prAM protocol P_{dec}. R constructs a co-nondeterministic circuit D as follows: On input a random string (r_1, r_2, r_3), which is interpreted as randomness for R_{pr}, randomness for A_{comp} and randomness for P_{dec}, respectively, D first runs $R_{\text{pr}}(1^n, (A_{\text{comp}}, P_{\text{dec}}); r_1)$ to obtain an input z of length ℓ with $n \leq \ell = O(\text{poly}(n))$. Then, using the fact that f is computable in polynomial time on a nondeterministic machine, D computes $f(z)$ using co-nondeterminism. Let A'_{comp} and P'_{dec} denote the versions of A_{comp} and P_{dec}, respectively, clocked to run in time t. Finally, the circuit D computes $A'_{\text{comp}}(z, r_2)$, co-nondeterministically verifies that there is no Merlin message that would lead P'_{dec} with input $A'_{\text{comp}}(z, r_2)$ and randomness r_3 to output $f(z)$, and accepts if and only if the verification succeeds.

Before moving further, we observe that, if the pair $(A_{\text{comp}}, P_{\text{dec}})$ is sound for f and obeys the running time and compression bounds, then D accepts at least a constant fraction of its inputs. This holds because for such n, $R_{\text{pr}}(1^n, (A_{\text{comp}}, P_{\text{dec}}); r_1)$ outputs, with probability at least a constant $\delta > 0$ over a random choice of r_1, an input z of length $\ell \geq n$ such that $(A_{\text{comp}}, P_{\text{dec}})$ fails to compute $f(z)$ with completeness $2/3$. For those z, it holds for a fraction of at least $1/3$ of strings (r_2, r_3) that there is no Merlin message that leads $P_{\text{dec}}(A_{\text{comp}}(z, r_2); r_3)$ to output $f(z)$. Thus, with probability at least $\delta' = \delta/3$, D accepts a triple (r_1, r_2, r_3).

After constructing D, R constructs a new co-nondeterministic circuit D' that is composed of k independent copies of D and that accepts if and only if at least one of the copies accepts, where k is constant to be set. R then computes $H(D')$, obtaining a set S of strings of the form $\rho = (\rho_1, \rho_2, \ldots, \rho_k)$, where each ρ_i is of the form (r_1, r_2, r_3). Finally, R outputs $R_{\text{pr}}(1^n, A_{\text{comp}}, P_{\text{dec}}; r_1)$ for all r_1 that appear in S. R always has an accepting computation path for every input since so does the generator H. Recall that if $(A_{\text{comp}}, P_{\text{dec}})$ is sound for f, then the acceptance probability of D is at least δ'. This means that the acceptance probability of D' is at least $1-(1-\delta')^k \geq 1-\exp(-\delta' k)$, which can be made at least $1/2$ by setting $k = \Theta(1/\delta)$. In this case, $H(D')$ outputs a hitting-set for D' on every accepting computation path. Let ρ be a string that hits D'. In that case, there must be some $\rho_i = (r_1, r_2, r_3)$ that hits D, which means that P_{dec} fails to compute $f(z)$ with perfect completeness on input $z = R_{\text{pr}}(1^n, A_{\text{comp}}, P_{\text{dec}}; r_1)$. As such a z is on the list output by R on every accepting computation path, R is a list-refuter for f against prAMTICOMP$[t(n), s(n)]$ protocols with promised soundness for f.

Let m denote the description length of $(A_{\text{comp}}, P_{\text{dec}})$. The co-nondeterministic circuit D' constructed by R on inputs 1^n and $(A_{\text{comp}}, P_{\text{dec}})$ has size $\text{poly}(m, n, t(\text{poly}(n))) = \text{poly}(m, t(\text{poly}(n)))$ since $t(n) \geq n$, and thus computing $H(D')$ takes time $T(\text{poly}(m, t(\text{poly}(n))))$. Finally, R needs to compute $R_{\text{pr}}(1^n, A_{\text{comp}}, P_{\text{dec}}; r_1)$ for at most $T(\text{poly}(m, t(\text{poly}(n))))$ strings r_1, and each such execution takes time $\text{poly}(m, n)$. The final running time is thus $T(\text{poly}(m, t(\text{poly}(n)))) + \text{poly}(m, n) = T(\text{poly}(m, t(\text{poly}(n))))$ since $T(n) \geq n$. □

We now exhibit a probabilistic polynomial-time refuter for the identity function against bottleneck protocols with imperfect completeness. The intuition is that strings z for which a bottleneck protocol computes identity correctly with completeness $2/3$ and soundness $1/3$ can be described succinctly via the output of the compression phase. Thus, the protocol must fail to compute identity for most z, as most z do not admit a succinct representation.

Proposition 39. *For every constant $\epsilon \in (0, 1)$, there exists a probabilistic polynomial-time refuter for the identity function against* prAMTICOMP$[\infty, n^\epsilon]$ *with completeness* $2/3$ *and soundness* $1/3$.

Proof. Fix $\epsilon \in (0, 1)$. The probabilistic polynomial-time refuter R_{pr}, on input 1^n and a pair $(A_{\text{comp}}, P_{\text{dec}})$ of description length m just outputs a random string z of length $\ell = \Theta(n)$ to be defined precisely in the next paragraph.

Assume that $(A_{\text{comp}}, P_{\text{dec}})$ computes the identity function with completeness $2/3$ and soundness $1/3$ on an input z of length ℓ, and that $|A_{\text{comp}}(z)| \leq \ell^\epsilon$. By an averaging argument, there exists a random sequence r_1 for A_{comp} such that the following property holds with probability at least $1/3$ over a random sequence r_2 for P_{dec}: Any reply from Merlin for the protocol $P_{\text{dec}}(A_{\text{comp}}(z; r_1); r_2)$ leads to either acceptance or the correct output z, and there exists a Merlin reply that leads to the correct output z. If we let $\pi_z = A_{\text{comp}}(z; r_1)$ and fix $(A_{\text{comp}}, P_{\text{dec}})$, we can describe z as one of the only three possible outputs of $P_{\text{dec}}(\pi_z)$ for which the property above holds. This description for z has length at most $\ell^\epsilon + c$ for some constant c, and thus at most $2^{\ell^\epsilon+c+1}$ out of the 2^ℓ strings of length ℓ can have such a short description. We then set $\ell = \max(n, n_0) = \Theta(n)$, where n_0 is the smallest integer such that $2^{n_0^\epsilon+c+1}/2^{n_0} \leq 1/3$. With ℓ as the output length, the probability that the refuter succeeds is at least $2/3$. □

For completeness, as we now have all the steps involved in Theorem 8, we tie them together in a formal proof.

Proof of Theorem 8. The implication $1 \implies 2$ holds trivially by taking the identity for f. The implication $2 \implies 3$ is Theorem 33. The implication $3 \implies 1$ follows by combining Theorem 38 and Proposition 39 with polynomial time bounds. □

A similar proof with bounds as in Theorem 34 establishes the following middle-of-the-road equivalence.

Theorem 40. *The following are equivalent:*

1. *For some constant $\epsilon \in (0,1)$, there exists a nondeterministic $2^{\text{polylog}(n)}$-time list-refuter for the identity function against* $\text{prAMTICOMP}[n \cdot 2^{(\log n)^\epsilon}, 2^{(\log n)^\epsilon}]$ *protocols with promised soundness for identity.*

2. *For some constants $a \geq 1$ and $\epsilon \in (0,1)$, there exists a function f computable in nondeterministic time $2^{(\log n)^a}$ that admits a nondeterministic polynomial-time list-refuter against* $\text{prAMTICOMP}[2^{(\log n)^{a+\epsilon}}, 2^{(\log n)^\epsilon}]$ *protocols with promised soundness for f.*

3. *There exists a targeted hitting-set generator that achieves the derandomization* $\text{prAM} \subseteq \text{NTIME}[2^{\text{polylog}(n)}]$.

6 Derandomization under uniform worst-case hardness

Our technique also leads to new results in the traditional uniform worst-case setting. Under worst-case hardness against probabilistic algorithms with non-adaptive oracle access to SAT, we obtain average-case derandomization results for prAM. Moreover, by further strengthening the hardness assumption, we may also conclude full (infinitely-often) derandomization of prAM. As previously mentioned, these results extend to average-case derandomization of $\text{prBPP}_{||}^{\text{SAT}}$.

6.1 Average-case simulation

In this section, we develop our average-case derandomization results for prAM under worst-case uniform hardness assumptions (where hardness is against $\text{BPTIME}_{||}^{\text{SAT}}$). Our results in this setting work as follows: Assume there exists a hard language $L \in \text{NTIME}[T(n)] \cap \text{coNTIME}[T(n)]$. To derandomize some prAM protocol P on input length n, we first consider the hard language L at some suitable input length ℓ, which depends on the hardness of L (for Theorem 10, for example, we take $\ell = \Theta(\log n)$). Then we let f be the function that maps any input $x \in \{0,1\}^n$ to the truth table of L at input length ℓ, and it follows from the complexity of L that f is computable in nondeterministic time $2^\ell c\dot{T}(\ell)$. Finally, we instantiate our targeted HSG construction H with f and use it to derandomize P.

For the reconstruction, we make use of the strong resilient soundness property of Corollary 30. If the average-case derandomization fails, to decide whether z of length ℓ is in L, we first sample multiple candidate "good" strings x that hopefully lead to a distinguisher D_x for the generator (enough so that we expect at least one "good" x with high probability). Then, we run the reconstruction for all of them, accepting if and only if at least one of those outputs 1. By the strong resilient soundness property and amplification, with high probability every execution either fails or outputs $f(x)_z = L(z)$, and in the high probability case that we sample at least one "good" x, some execution outputs $L(z)$, meaning we can compute L efficiently on input length ℓ.

First, we present such a result at the high end of the derandomization spectrum.

Theorem 41 (Strengthening of Theorem 10). *If* $\mathrm{NTIME}[2^{an}] \cap \mathrm{coNTIME}[2^{an}]$ *is not included in* $\mathrm{BPTIME}[2^{(\log(a+1))^2 n}]^{\mathrm{SAT}}_{||}$ *for some constant $a > 0$, then for all $e > 0$ it holds that*

$$\mathrm{prAM} \subseteq \text{io-Heur}_{1/n^e}\mathrm{NP}$$
$$\mathrm{prBPP}^{\mathrm{SAT}}_{||} \subseteq \text{io-Heur}_{1/n^e}\mathrm{P}^{\mathrm{SAT}}_{||}.$$

Proof. We first argue the result for prAM. Consider derandomizing a prAM protocol P_Π for a problem Π running in time $O(n^k)$ for some constant k. Let S be an $O(n^s)$-time sampler for a distribution in $\{0,1\}^n$ and e be a constant such that we want to "fool" S with probability at least $1 - 1/n^e$. Let f be a function mapping every $z \in \{0,1\}^n$ to the truth table of the hard language $L \in \mathrm{NTIME}[2^{an}] \cap \mathrm{coNTIME}[2^{an}]$ at input length $\ell = \ell(n) = \Theta(\log n)$ to be set precisely later. Note that f is computable in nondeterministic time $T(n) = 2^{(a+1)\ell}$. Instantiate the generator H of Corollary 30 with f, run H on input $z = 0^n$ (recall f maps every string in $\{0,1\}^n$ to the same truth table) and co-nondeterministic circuit size $m = O(n^{2k})$, and use it to attempt to derandomize P_Π in nondeterministic time $\mathrm{poly}(T(n), n^{2k}) = \mathrm{poly}(n)$.

If the derandomization fails for almost-all input lengths, even heuristically, then for almost-all input lengths n, $S(1^n)$ outputs with probability at least $1/n^e$ a string $x \in \{0,1\}^n$ such that the simulation errs on x, i.e., the circuit D_x obtained from x and P_Π using Proposition 21 is a distinguisher for $H(0^n, D_x)$. To compute L at input length ℓ, it then suffices to do the following: On input $w \in \{0,1\}^\ell$, first use S to sample $t = \Theta(n^e)$ inputs $x_1, \ldots, x_t$ and use these to construct a list $D_{x_1}, \ldots, D_{x_t}$ of candidate distinguishers for $H(0^n, D_x)$. With high probability, this list contains an actual distinguisher for the generator. Let A_{rec} be the algorithm of Corollary 30, amplified by parallel repetition to have negligible soundness 2^{-n}, i.e., with probability at least $1 - 2^n$, the algorithm outputs either $\perp$ or a correct evaluation of f. Finally, run A_{rec} with inputs 0^n, index w (recall $f(0^n)$ equals the truth table of L at input length ℓ) and D_{x_i} for every sampled input x_i, and accept if and only if some execution outputs 1. To see that this is correct, note that by a union bound, with high probability every execution of A_{rec} is successful in the sense that it either outputs $f(0^n)_w = L(w)$ or $\perp$. Conditioned on there being a distinguisher in the list, we are guaranteed to output the correct value of $L(w)$ with high probability.

The running time for the reconstruction is $O(n^{e+s})$ for generating the $t = \Theta(n^e)$ samples, and $O(n^{2k})^{O((\log r)^2)}$ per sample for running A_{rec}, where $r = O(((a+1)\ell)/(k \log n))$, for a total of $O(n^e(n^s + n^{O(k(\log r)^2)}))$. By setting $\ell = dk \log n$, we have that $r = O(d(a+1))$ and we can upper bound the total running time by $n^{O(e+s+k(\log(d(a+1)))^2)}$. In terms of the input length ℓ, this is $2^{(\log(a+1))^2 \ell}$ when d is a sufficiently large constant depending on a, e, s. This concludes the argument for prAM.

Now, we argue the result for $\mathrm{prBPP}_{||}^{\mathrm{SAT}}$. To do so, we use the containment $\mathrm{prBPP}_{||}^{\mathrm{SAT}} \subseteq \mathrm{P}_{||}^{\mathrm{prAM}}$ [CR11]. It suffices to show that every deterministic polynomial-time algorithm with non-adaptive oracle access to a paddable prAM-complete problem $\Gamma \in \mathrm{prAMTIME}[n]$ can be simulated by deterministic polynomial-time algorithms with non-adaptive oracle access to SAT. Let M be a deterministic algorithm with non-adaptive oracle access to Γ running in time $O(n^b)$ and S be an $O(n^s)$-time sampler that we want to "fool" with probability at least $1-1/n^e$. Since Γ is paddable, we may assume that every query made by M on inputs of length n is of length $O(n^b)$ (at the expense of increasing its running time to $O(n^{2b})$). To simulate M on an input x of length n, let f be a function mapping every $z \in \{0,1\}^n$ to the truth table of L at input length $\ell = \ell(n) = \Theta(\log n)$. As before, f is computable in nondeterministic time $2^{(a+1)\ell}$. Instantiate the generator H of Corollary 30 with f and input $z = 0^n$ and use it to derandomize Γ at input length $O(n^b)$ in order to obtain a $\mathrm{P}_{||}^{\mathrm{SAT}}$ simulation for M. Whenever M with input x queries Γ, we instead query the SAT oracle whether the nondeterministic simulation of Γ using H with input 0^n and co-nondeterministic circuit size $m = O(n^{2b})$ accepts. This simulation runs in $\mathrm{P}_{||}^{\mathrm{SAT}}$ since M is non-adaptive.

If this derandomization fails on almost-all input lengths n, then as before we can use S to sample $t = \Theta(n^e)$ inputs $x_1, \ldots, x_t$ such that with high probability the simulation fails on some x_i. Let $Q(M,x)$ be the set of queries to Γ made by M on input x. If the simulation fails on x_i, it must be the case that some query q in $Q(M,x_i)$ (and also in the promise of Γ) was answered incorrectly. Since the protocol for Γ has perfect completeness, it must be the case that $q \in \Pi_N$ and that D_q is a distinguisher for $H(0^n, D_q)$. The reconstruction is as before though we use the sets $Q(M,x_i)$ for $i \in [t]$ to obtain the list of candidate generators, and correctness follows by the same argument as in the prAM case. The running time analysis is similar to the one for the case of prAM. ☐

At the low end, we are able to obtain a slightly stronger average-case derandomization result. Instead of having a different simulation for each sampler, we obtain a single simulation (depending on the problem in $\mathrm{prAM}/\mathrm{prBPP}_{||}^{\mathrm{SAT}}$ and the constant ϵ) that "fools" every polynomial-time sampler.

Theorem 42. *If* $\mathrm{NEXP} \cap \mathrm{coNEXP} \not\subseteq \mathrm{BPTIME}[n^{b((\log n)^2)}]_{||}^{\mathrm{SAT}}$ *for all* $b > 0$*, then for every* $\epsilon > 0$ *and all* $e > 0$

$$\mathrm{prAM} \subseteq \text{io-}\mathrm{Heur}_{1/n^e}\mathrm{NTIME}[2^{n^\epsilon}]$$
$$\mathrm{prBPP}_{||}^{\mathrm{SAT}} \subseteq \text{io-}\mathrm{Heur}_{1/n^e}\mathrm{DTIME}[2^{n^\epsilon}]_{||}^{\mathrm{SAT}}.$$

Moreover, for any Π *in* prAM *or* $\mathrm{prBPP}_{||}^{\mathrm{SAT}}$ *and* $\epsilon > 0$*, there is a single simulation that works for all* $e > 0$*.*

Proof. We begin with the argument for prAM. Let L be a hard language in $\mathrm{NTIME}[2^{n^a}] \cap \mathrm{coNTIME}[2^{n^a}]$ for some constant $a \geq 1$. Consider derandomizing a protocol P_Π for a problem $\Pi \in \mathrm{prAMTIME}[n^k]$ for constant k. Let $\epsilon > 0$ and f be the function mapping every $z \in \{0,1\}^n$ to the truth table of L at input length $\ell = n^\epsilon$. Note that f is computable in nondeterministic time $T(n) = 2^{n^{a\epsilon}}$. Instantiate the generator H of Corollary 30 with f, run H on input $z = 0^n$ and co-nondeterministic circuit size $m = O(n^{2k})$, and use it to derandomize P_Π. The simulation runs in nondeterministic time $\mathrm{poly}(T(n), n^{2k})$, which is at most $2^{n^{\epsilon'}}$ for any $\epsilon' > 0$ by taking a sufficiently small $\epsilon > 0$.

The reconstruction is identical to that of Theorem 41 but with $\ell = n^\epsilon$. The running time is $O(n^{e+s})$ to generate the samples and $(n^{2k})^{O((\log r)^2)}$ per sample for running A_{rec}, where $r =$

$O(\log(T(n))/\log n)$, for a total of $O(n^e(n^s + n^{O((\log r)^2)}))$. Given our parameter choices, $r = O(n^{a\epsilon})$, and the expression is upper bounded by $O(n^e(n^s + n^{O((a\epsilon \log n)^2)}))$. As the input length is $\ell = n^\epsilon$ for constant ϵ, there exists a constant b (depending on a, e, s, ϵ) such that the running time is upper bounded by $\ell^{b(\log n)^2}$. If hardness holds for all $b > 0$, then the same simulation works for any constant value of s and e, i.e., for any polynomial-time sampler and any inverse-polynomial error probability.

The proof for $\mathrm{prBPP}_{||}^{\mathrm{SAT}}$ is also almost identical to that of Theorem 41, where we derandomize the "oracle" Γ using the generator H from Corollary 30 instantiated with the function f that maps every $z \in \{0,1\}^n$ to the truth table of L at input length $\ell = n^\epsilon$ and use a set of queries instead of a set of inputs to obtain the list of candidate distinguishers for the reconstruction. This approach naturally leads to a simulation in $\mathrm{P}_{||}^{\mathrm{NTIME}[2^{n^\epsilon}]}$, and we obtain the $\mathrm{DTIME}[2^{n^\epsilon}]_{||}^{\mathrm{SAT}}$ simulation by replacing the original queries with padded SAT queries. $\square$

6.2 Infinitely-often all-input simulation

By introducing nondeterminism in the algorithms we require hardness for, we are able to extend Theorem 10 to conclude full (infinitely-often) derandomization of prAM. We have shown that, if the HSG construction of Theorem 10 fails to obtain average-case derandomization of prAM, then we are able to efficiently sample candidate distinguishers with the hope that at least one is "good". However, if the HSG fails in the worst case, it is harder to pinpoint exactly where it does so as to obtain a distinguisher. To solve this, we have Merlin send a "good" input x. This necessitates a lower bound against $\mathrm{MATIME}_{||}^{\mathrm{SAT}}$, but allows for concluding full (infinitely-often) derandomization of prAM and $\mathrm{prBPP}_{||}^{\mathrm{SAT}}$.

Theorem 43. *If* $\mathrm{NTIME}[2^{an}] \cap \mathrm{coNTIME}[2^{an}] \not\subseteq \mathrm{MATIME}[2^{(\log(a+1))^2 n)}]_{||}^{\mathrm{SAT}}$ *for some constant* $a > 0$*, then*

$$\mathrm{prAM} \subseteq \text{io-NP}$$
$$\mathrm{prBPP}_{||}^{\mathrm{SAT}} \subseteq \text{io-P}_{||}^{\mathrm{SAT}}.$$

Proof. We argue the result for prAM first. Let $\Pi \in \mathrm{prAMTIME}[n^k]$ for some constant k and let L be a hard language in $\mathrm{NTIME}[2^{an}] \cap \mathrm{coNTIME}[2^{an}]$. Let f be a function mapping every input $z \in \{0,1\}^n$ to the truth table of L at input length $\ell = \Theta(\log n)$ to be set precisely later. Note that f is computable in nondeterministic time $T(n) = 2^{(a+1)\ell}$. Instantiate the generator H of Corollary 30 with f, run H on input $z = 0^n$ and co-nondeterministic circuit size $m = O(n^{2k})$, and use it to derandomize Π in time $\mathrm{poly}(T(n), n) = \mathrm{poly}(n)$.

If the simulation fails for some input of almost-all input lengths, then for almost-all input lengths n there exists an $x \in \Pi_N$ of length n such that the simulation errs on x, i.e., the circuit D_x of Proposition 21 instantiated with the protocol for Π and x is a distinguisher for $H(0^n, D_x)$. Let A_{rec} be the reconstructor of Corollary 30 and consider the following Merlin-Arthur protocol for L, where the protocol has parallel oracle access to SAT: On input $w \in \{0,1\}^\ell$, Merlin sends x, and Arthur runs $A_{\mathrm{rec}}(0^n, D_x)$ to compute the w-th bit of $f(0^n) = L(w)$. If A_{rec} outputs $\perp$, then the protocol rejects, otherwise, it accepts if and only if A_{rec} outputs 1. Because A_{rec} is a probabilistic algorithm with parallel access to an oracle for SAT, Arthur can select the randomness required for it and then run the underlying deterministic parallel-SAT-oracle computation, meaning this is indeed a $\mathrm{MA}_{||}^{\mathrm{SAT}}$ protocol. Completeness follows since Merlin can send a correct value of x, and soundness follows from the strong resilience property of A_{rec}: Even if Merlin sends a "bad" x', A_{rec} is still guaranteed to either fail or output $L(w)$ with high probability.

To finish the argument for prAM, note that the running time of the protocol is just the running time of A_{rec}, which is $\mathrm{poly}(n)\cdot(m\cdot\log T(n))^{O((\log r)^2)}$ for $r = O(\log(T(n))/\log m)$. Since $m = O(n^{2k})$ and setting $\ell = dk\log n$, we have $r = O(d(a+1))$ and the running time for the protocol is upper bounded by $n^{O(k(\log(d(a+1)))^2)}$. In terms of the input length ℓ, this is $2^{(\log(a+1))^2\ell)}$ when d is a sufficiently large constant depending on a.

The simulation for $\mathrm{prBPP}_{||}^{\mathrm{SAT}}$ is similar to before and the reconstruction is identical to the prAM case: If the simulation fails, then there is a query q of length $O(n^k)$ (which results in a distinguisher of size $O(n^{2k})$) that Merlin can send Arthur to make Arthur output $L(w)$ with high probability. Soundness also follows exactly as in the prAM case and the running time is again $2^{(\log(a+1))^2\ell)}$. $\square$

We only state the previous result for the high-end parameter setting because stronger results are already known for the low end. For example, to conclude a subexponential derandomization of prAM, it suffices for there to exist a language in NEXP $\cap$ coNEXP that is hard for a subclass of $\mathrm{MA}_{||}^{\mathrm{SAT}}$ [AvM17]. In comparison with ours, other results that conclude the same derandomization either require hardness of nondeterministic algorithms against much larger deterministic time bounds, e.g., $\mathrm{NE} \cap \mathrm{coNE} \not\subseteq \mathrm{DTIME}[2^{2^{n^{\epsilon}}}]$ for some $\epsilon > 0$ [IKW02] or hardness of deterministic algorithms against slightly less space, e.g., $\mathrm{E} \not\subseteq \mathrm{SPACE}[2^{\epsilon n}]$ for some $\epsilon > 0$ [Lu01].

7 Unconditional mild derandomization

In this section, we establish our unconditional mild derandomization result for prAM and extend it to $\mathrm{prBPP}_{||}^{\mathrm{SAT}}$. We employ a similar win-win argument to that of the proof of Theorem 9: Either some hardness assumption/class separation holds (here, $\Sigma_2\mathrm{EXP} \not\subseteq \mathrm{NP/poly}$), in which case we get derandomization right away. Or else we get a complexity collapse which we can use to construct a hard function f that has the efficiency requirements we need to apply one of our targeted hitting-set constructions (in this case Theorem 42, which requires hardness against $\mathrm{BPTIME}[2^{\mathrm{polylog}(n)}]_{||}^{\mathrm{SAT}}$).

As a first step toward the win-win argument, we prove an "easy-witness lemma" for $\Sigma_2\mathrm{EXP}$, which allows for the collapse $\mathrm{P}^{\Sigma_2\mathrm{EXP}} \subseteq \mathrm{EXP}$ from the assumption that $\Sigma_2\mathrm{EXP} \subseteq \mathrm{NP/poly}$. Then we consider two cases:

- $\Sigma_2\mathrm{EXP} \not\subseteq \mathrm{NP/poly}$. In this case, the derandomization result follows from standard hardness vs. randomness tradeoffs.
- $\Sigma_2\mathrm{EXP} \subseteq \mathrm{NP/poly}$. In this case, we diagonalize against $\mathrm{BPTIME}[2^{\mathrm{polylog}(n)}]_{||}^{\mathrm{SAT}}$ in $\mathrm{P}^{\Sigma_2\mathrm{EXP}} = \mathrm{EXP}$, and then instantiate Theorem 42 to conclude the proof.

To diagonalize against $\mathrm{BPTIME}[2^{\mathrm{polylog}(n)}]_{||}^{\mathrm{SAT}}$, we make use of the inclusion $\mathrm{prBPP}_{||}^{\mathrm{SAT}} \subseteq \mathrm{P}_{||}^{\mathrm{prAM}}$ and diagonalize against deterministic algorithms with non-adaptive oracle access to prAM instead.

7.1 Nondeterministic easy witnesses

In this section, we prove our "easy witness lemma" for $\Sigma_2\mathrm{EXP}$. One way of thinking of Σ_2 computations is as follows: On input x, guess a string y and then run a co-nondeterministic verifier on input (x, y). This view allows us to abstract the co-nondeterministic verification and think of y as a witness for x. In this section, we show that if $\Sigma_2\mathrm{EXP} \subseteq \mathrm{NP/poly}$, then every language in $\Sigma_2\mathrm{EXP}$ has witnesses that are the truth tables of functions computed by polynomial-size single-valued circuits. To do so, we use the following result to convert hardness against single-valued circuits into hitting sets for co-nondeterministic circuits.

Lemma 44 ([Uma03]). *There is a universal constant b and a deterministic polynomial-time algorithm that, on input 1^m and a truth table y of a function with single-valued circuit complexity at least m^b, outputs a set S of size $O(|y|^b)$ that hits co-nondeterministic circuits of size m that accept at least half of their inputs.*

We also need the following equivalence from [AvM17].

Lemma 45 ([AvM17]). $\Sigma_2\mathrm{EXP} \not\subseteq \mathrm{NP/poly}$ *if and only if* $\mathrm{prAM} \subseteq \text{io-}\Sigma_2\mathrm{TIME}[2^{n^\epsilon}]/n^\epsilon$ *for all* $\epsilon > 0$.

We are now ready to prove our easy witness result for $\Sigma_2\mathrm{EXP}$.

Theorem 46. *Assume* $\Sigma_2\mathrm{EXP} \subseteq \mathrm{NP/poly}$*. Then* $\Sigma_2\mathrm{EXP}$ *has single-valued witnesses of polynomial size, i.e., for every* $L \in \Sigma_2\mathrm{EXP}$ *and linear-time (in its input length) co-nondeterministic verifier* H *for* L*, the following holds: For every* $x \in L$*, there exists a single-valued circuit* C_x *of size* $\mathrm{poly}(|x|)$ *such that* $H(x, \cdot)$ *accepts the exponential-length truth table of* C_x*.*

Proof. We show that $\Sigma_2\mathrm{E}$ has single-valued witness circuits of size n^c for some constant c. The result for $\Sigma_2\mathrm{EXP}$ then follows by padding.

Assume that $\Sigma_2\mathrm{E}$ does not have single-valued witness circuits of size n^c for any constant c. This implies that for all $c \geq 1$, there is a co-nondeterministic verifier H_c that takes as input a string x and a string y of length $2^{O(|x|)}$, runs in time $2^{O(|x|)}$, and has the following property: For infinitely many n, there is a input x' of length n such that $H_c(x', y')$ accepts for some y', but *every* y accepted by $H_c(x', \cdot)$ has single-valued circuit complexity at least n^c. Thus, there are infinitely many n such that, if we give x' as n bits of advice, guess a string y of length $2^{O(n)}$, and verify that $H_c(x', y)$ accepts (using co-nondeterminism), we are guaranteed that y encodes the truth table of a function with single-valued circuit complexity at least n^c. This gives us a Σ_2-procedure for obtaining hard functions, which we use to derandomize prAM and obtain a contradiction to Lemma 45.

Let $\Pi \in \mathrm{prAM}$ and let P_Π be a protocol for Π that runs in time $O(\ell^a)$ on input length ℓ. By Proposition 21, to derandomize P_Π it suffices to have a set S that hits any co-nondeterministic circuit of size $O(\ell^{2a})$ that accepts at least half of its inputs. To obtain such a set using Lemma 44, we need to first obtain a truth table of single-valued circuit complexity at least $\Omega(\ell^{2ab})$, where b is the constant from the lemma. Recall that our objective is to obtain a subexponential (time 2^{n^ϵ} for all $\epsilon > 0$) simulation. To this end, let $\epsilon > 0$ be sufficiently small and consider the verifier H_c for $c = 3ab/\epsilon$ on inputs of length $n = \ell^\epsilon$. If n is one of the infinitely many input lengths for which there exists x' such that every string accepted by $H_c(x', \cdot)$ has single-valued circuit complexity at least $n^c = \ell^{3ab}$, then we can obtain such a hard string by having x' as advice, guessing $y \in \{0,1\}^{2^{O(\ell^\epsilon)}}$ and verifying that $H_c(x', y)$ accepts.

In parallel, apply Lemma 44 to y to obtain a set S of size $2^{O(\ell^\epsilon)}$, and use S to derandomize the prAM computation (guessing a Merlin response for each string in S). Finally, accept if and only if both $H_c(x', y)$ and the prAM simulation accept. All of this can be carried out in $\Sigma_2\mathrm{TIME}[2^{O(\ell^\epsilon)}]/\ell^\epsilon$. Since ϵ is an arbitrarily small constant and the simulation works for infinitely many input lengths ℓ, we obtain a contradiction to Lemma 45. □

Theorem 46 allows us to establish the following complexity class collapse in case $\Sigma_2\mathrm{EXP} \subseteq \mathrm{NP/poly}$. The corollary represents the role our easy witness result plays in the proof of Theorem 11.

Corollary 47. *If* $\Sigma_2\mathrm{EXP} \subseteq \mathrm{NP/poly}$*, then* $\mathrm{P}^{\Sigma_2\mathrm{EXP}} = \mathrm{EXP}$.

Proof. Under the hypothesis from the statement, we show that $\Sigma_2\mathrm{EXP} = \mathrm{coNEXP}$, which suffices by combining Lemma 45 and Lemma 37. The hypothesis and Lemma 45 guarantee the negation of $\mathrm{prAM} \subseteq \text{io-}\Sigma_2\mathrm{TIME}[2^{n^\epsilon}]/n^\epsilon$ for all ϵ, which in turn implies the negation of $\mathrm{prAM} \subseteq \text{io-NTIME}[2^{n^\epsilon}]/n^\epsilon$ for all ϵ, and thus the contrapositive of Lemma 37 implies EXP = NEXP and therefore $\Sigma_2\mathrm{EXP} = \mathrm{coNEXP} = \mathrm{EXP}$. Finally, we have $\mathrm{P}^{\Sigma_2\mathrm{EXP}} = \mathrm{P}^{\mathrm{EXP}} = \mathrm{EXP}$.

To show that $\Sigma_2\mathrm{EXP} = \mathrm{coNEXP}$ it suffices by padding to show that every $L \in \Sigma_2\mathrm{E}$ is in coNEXP. Fix $L \in \Sigma_2\mathrm{E}$. By Theorem 46, L has single-valued witnesses of size n^c for some constant c. On input $x \in \{0,1\}^n$, we cycle through all nondeterministic circuits C of size n^c and compute their truth tables in time $O(2^{n^c})$. For each truth table T, we then run $V(x,T)$ (where V is a co-nondeterministic verifier for L), accepting if and only if some verification accepts. All of this runs in exponential co-nondeterministic time, so we are done. ☐

7.2 Simulation

We now execute our win-win strategy and establish Theorem 11 and its strengthening for $\mathrm{prBPP}_{||}^{\mathrm{SAT}}$ in lieu of prAM. We first consider the case where $\Sigma_2\mathrm{EXP} \not\subseteq \mathrm{NP/poly}$. In this case simulations of the required type that work on all inputs of a given length are provided by Lemma 45 for prAM. We argue the same simulations follow for $\mathrm{prBPP}_{||}^{\mathrm{SAT}}$.

Lemma 48. *If* $\Sigma_2\mathrm{EXP} \not\subseteq \mathrm{NP/poly}$*, then for every* $\epsilon > 0$

$$\mathrm{prBPP}_{||}^{\mathrm{SAT}} \subseteq \text{io-}\Sigma_2\mathrm{TIME}[2^{n^\epsilon}]/n^\epsilon.$$

Proof. We use the inclusion $\mathrm{prBPP}_{||}^{\mathrm{SAT}} \subseteq \mathrm{P}_{||}^{\mathrm{prAM}}$. Let k be a constant and M be an $O(n^k)$-time deterministic machine with non-adaptive oracle access to a paddable prAM-complete problem $\Gamma \in \mathrm{prAMTIME}[n]$. We assume that all queries made by M on inputs of length n are of length $O(n^k)$ at the expense of increasing M's running time to $O(n^{2k})$.

Our approach is to use Lemma 44 to derandomize the queries made to Γ while making sure that the overall simulation of M can be carried out in subexponential Σ_2-time. To derandomize Γ at input length $O(n^k)$ using the lemma, we need to obtain a truth table of single-valued circuit complexity at least $\Omega(n^{2bk})$, where b is the constant from the lemma. Let $\epsilon > 0$ and $L \in \Sigma_2\mathrm{E}$ be a language that has nondeterministic circuit complexity at least $n^{3bk/\epsilon}$ for infinitely many input lengths (which is guaranteed to exist by the hypothesis of the theorem). The simulation of M on inputs x goes as follows: Given as advice the number of strings of length n^ϵ that are in L, the Σ_2 algorithm guesses the truth table of L at input length n^ϵ, verifies it, and uses it as the string y in Lemma 44. More precisely, after guessing the truth table, the algorithm performs the following operations in parallel:

- It uses an existential and a universal guess to verify that the guessed truth table for L is correct. This is possible because the algorithm has as advice the number of strings of length n^ϵ that are in L, and thus it can existentially guess which strings are in L and only verify those, with the guarantee that the others are not in L.

∘ It guesses which of the queries to Γ that M makes on input x are answered positively and which are answered negatively. For each query that is guessed to be answered positively, it uses the set S from Lemma 44 and the existential phase to verify that there is a random-bit string in S for which Merlin can provide a witness. Similarly, it uses S and the universal phase to verify each query that is guessed to be answered negatively.

We note that the only existential computation paths that survive the computation are the ones where the truth table of L at input length n^ϵ was guessed correctly. In this case, and in the case that n^ϵ is one of the infinitely many input lengths where L has nondeterministic circuit complexity at least $n^{3bk/\epsilon}$, it holds that the guessed truth table has high enough (single-valued) nondeterministic circuit complexity such that S hits the co-nondeterministic circuits given by Proposition 21 for negative instances of Γ at input length $O(n^k)$. This further guarantees that the surviving existential computation paths are those that correctly guess the answers to all queries M makes on input x that are in the promise of Γ. This suffices to obtain a simulation of M that is correct on infinitely many input lengths since M is insensitive to variations in answers to queries that are outside the promise (even when the same query is answered differently on different occasions). Finally, we note that the entire procedure runs in time $2^{O(n^\epsilon)}$, which can be made smaller than $2^{n^{\epsilon'}}$ for any $\epsilon' > 0$ by taking ϵ to be sufficiently small. □

The other case of the win-win analysis is when $\Sigma_2\mathrm{EXP} \subseteq \mathrm{NP/poly}$. In this case, we use the collapse $\mathrm{P}^{\Sigma_2\mathrm{EXP}} = \mathrm{EXP}$ given by Corollary 47 and our targeted hitting-generator construction to obtain the desired simulation. We conclude:

Theorem 49 (Strengthening of Theorem 11). *For every $\epsilon > 0$ and every $e > 0$*

$$\mathrm{prBPP}_{||}^{\mathrm{SAT}} \subseteq \text{io-}\mathrm{Heur}_{1/n^e}\Sigma_2\mathrm{TIME}[2^{n^\epsilon}]/n^\epsilon.$$

Proof. If $\Sigma_2\mathrm{EXP} \not\subseteq \mathrm{NP/poly}$, then it follows that $\mathrm{prBPP}_{||}^{\mathrm{SAT}} \subseteq \Sigma_2\mathrm{TIME}[2^{n^\epsilon}]/n^\epsilon$ for all $\epsilon > 0$ by Lemma 48. Otherwise, by Corollary 47, we have that $\mathrm{P}^{\Sigma_2\mathrm{EXP}} = \mathrm{EXP}$. By Theorem 41, all we need to show is that $\mathrm{P}^{\Sigma_2\mathrm{EXP}} \not\subseteq \bigcup_{b\in\mathbb{N}} \mathrm{BPTIME}[n^{b((\log n)^2)}]_{||}^{\mathrm{SAT}}$. Given the containment $\mathrm{prBPP}_{||}^{\mathrm{SAT}} \subseteq \mathrm{P}_{||}^{\mathrm{prAM}}$ and a padding argument, it follows that $\bigcup_{b\in\mathbb{N}} \mathrm{BPTIME}[n^{b((\log n)^2)}]_{||}^{\mathrm{SAT}} \subseteq \mathrm{DTIME}[2^{\mathrm{polylog}(n)}]_{||}^{\mathrm{prAM}}$. It remains to show that $\mathrm{P}^{\Sigma_2\mathrm{EXP}} \not\subseteq \mathrm{DTIME}[2^{\mathrm{polylog}(n)}]_{||}^{\mathrm{prAM}}$, which we do by diagonalization.

Fix a prAM-complete problem Γ and note that if $L \in \mathrm{DTIME}[2^{\mathrm{polylog}(n)}]_{||}^{\mathrm{prAM}}$, then there exists a Turing machine M running in time $2^{\mathrm{polylog}(n)}$ with non-adaptive oracle access to Γ that computes L. Thus, it suffices to diagonalize against such machines with Γ as an oracle. Let S be the following $\Sigma_2\mathrm{EXP}$-oracle machine: On input $x \in \{0,1\}^n$, interpret x as a non-adaptive oracle Turing machine M_x with an oracle for Γ. Then, using binary search and the $\Sigma_2\mathrm{EXP}$ oracle, compute the number q of queries that M_x on input x makes that are answered negatively, where we let M_x run for at most 2^n steps. This is possible in $\mathrm{P}^{\Sigma_2\mathrm{EXP}}$ because $\mathrm{prAM} \subseteq \Pi_2\mathrm{P}$, so we can verify negative instances in $\Sigma_2\mathrm{EXP}$. Once we know q, we can simulate $M_x(x)$ for at most 2^n steps in $\Sigma_2\mathrm{EXP}$ as follows: Guess which q queries are negative and verify them in $\Sigma_2\mathrm{EXP}$ (again using the fact that $\mathrm{prAM} \subseteq \Pi_2\mathrm{P}$); then assume that the other queries are answered positively and simulate $M_x(x)$ directly with these answers. By querying the $\Sigma_2\mathrm{EXP}$ oracle S then outputs the opposite of this simulation. By construction, the language of S is in $\mathrm{P}^{\Sigma_2\mathrm{EXP}} \setminus \mathrm{DTIME}[2^{\mathrm{polylog}(n)}]_{||}^{\mathrm{prAM}}$. □

This concludes our discussion of the byproducts of our main results.

Acknowledgments

We thank Ronen Shaltiel and Chris Umans for answering questions about their work, Oded Goldreich for helpful feedback on the write-up, and Lijie Chen for suggesting the potential use of PCPs during a presentation of our preliminary results. An earlier version of part of this work appeared as [vMMS23].

References

[AvM17] Barış Aydınlıoğlu and Dieter van Melkebeek. Nondeterministic circuit lower bounds from mildly derandomizing Arthur-Merlin games. *Computational Complexity*, 26(1):79–118, 2017.

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

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

[BSGH$^+$05] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Short PCPs verifiable in polylogarithmic time. In *Conference on Computational Complexity (CCC)*, pages 120–134, 2005.

[CR11] Venkatesan T. Chakaravarthy and Sambuddha Roy. Arthur and Merlin as oracles. *Computational Complexity*, 20(3):505–558, 2011.

[CRT22] L. Chen, R. D. Rothblum, and R. Tell. Unstructured hardness to average-case randomness. In *Symposium on Foundations of Computer Science (FOCS)*, pages 429–437, 2022.

[CRTY20] Lijie Chen, Ron D. Rothblum, Roei Tell, and Eylon Yogev. On exponential-time hypotheses, derandomization, and circuit lower bounds: Extended abstract. In *Symposium on Foundations of Computer Science (FOCS)*, pages 13–23, 2020.

[CT21] Lijie Chen and Roei Tell. Hardness vs randomness, revised: Uniform, non-black-box, and instance-wise. In *Symposium on Foundations of Computer Science (FOCS)*, pages 125–136, 2021.

[CTW23] Lijie Chen, Roei Tell, and Ryan Williams. Derandomization vs refutation: A unified framework for characterizing derandomization. In *Symposium on Foundations of Computer Science (FOCS)*, pages 1008–1047, 2023.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *Journal of the ACM*, 62(4), 2015.

[Gol11] Oded Goldreich. In a world of P=BPP. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 191–232.

Springer, 2011. Part of the Lecture Notes in Computer Science book series (LNCS, volume 6650).

[Gol18] Oded Goldreich. On doubly-efficient interactive proof systems. *Foundations and Trends in Theoretical Computer Science*, 13:157–246, 2018.

[Gol20] Oded Goldreich. Two comments on targeted canonical derandomizers. In *Computational Complexity and Property Testing: On the Interplay Between Randomness and Computation*, pages 24–35. Springer, 2020.

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

[IKW02] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. *Journal of Computer and System Sciences*, 65(4):672 – 694, 2002.

[IW97] Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Symposium on Theory of Computing (STOC)*, page 220–229, 1997.

[IW01] Russell Impagliazzo and Avi Wigderson. Randomness vs time: Derandomization under a uniform assumption. *Journal of Computer and System Sciences*, 63(4):672–688, 2001.

[Jus76] J. Justesen. On the complexity of decoding reed-solomon codes (corresp.). *IEEE Transactions on Information Theory*, 22(2):237–238, 1976.

[Kor22] Oliver Korten. Derandomization from time-space tradeoffs. In *Computational Complexity Conference (CCC)*, pages 37:1–37:26, 2022.

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

[LP22] Yanyi Liu and Rafael Pass. Characterizing derandomization through hardness of Levin-Kolmogorov complexity. In *Computational Complexity Conference (CCC)*, volume 234, pages 35:1–35:17, 2022.

[LP23] Yanyi Liu and Rafael Pass. Leakage-Resilient Hardness vs Randomness. In *Computational Complexity Conference (CCC)*, volume 264, pages 32:1–32:20, 2023.

[Lu01] Chi-Jen Lu. Derandomizing Arthur-Merlin games under uniform assumptions. *Computational Complexity*, 10(3):247–259, 2001.

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

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

[Spi96] D.A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1731, 1996.

[SU05] Ronen Shaltiel and Christopher Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *Journal of the ACM*, 52(2):172–216, 2005.

[SU09] Ronen Shaltiel and Christopher Umans. Low-end uniform hardness versus randomness tradeoffs for AM. *SIAM Journal on Computing*, 39(3):1006–1037, 2009.

[TV07] Luca Trevisan and Salil Vadhan. Pseudorandomness and average-case complexity via uniform reductions. *Computational Complexity*, 16(4):331–364, 2007.

[Uma03] Christopher Umans. Pseudo-random generators for all hardnesses. *Journal of Computer and System Sciences*, 67(2):419 – 440, 2003.

[vMMS23] Dieter van Melkebeek and Nicollas Mocelin Sdroievski. Instance-wise hardness versus randomness tradeoffs for Arthur-Merlin protocols. In *Computational Complexity Conference (CCC)*, volume 264, pages 17:1–17:36, 2023.

[Wil16] R. Ryan Williams. Natural proofs versus derandomization. *SIAM Journal on Computing*, 45(2):497–529, 2016.