Refutation versus Derandomization for Arthur-Merlin Protocols

Dieter van Melkebeek
University of Wisconsin-Madison
dieter@cs.wisc.edu

Nicollas Mocelin Sdroievski
University of Wisconsin-Madison
sdroievski@wisc.edu

May 10, 2024

Abstract

Blackbox derandomization of probabilistic decision algorithms (the class BPP) or Arthur-Merlin protocols (the class AM) work through the construction of pseudo-random generators or hitting-set generators. Their existence is well-known to be equivalent to lower bounds for decision problems against circuits or nondeterministic circuits.

Whitebox derandomization in the BPP setting has been shown equivalent to the existence of targeted pseudo-random generators (Goldreich, LNCS). Chen and Tell (FOCS'21) established a near-equivalence with lower bounds for multi-bit functions against algorithms on almost-all inputs. Recently, Chen, Tell and Williams (FOCS'23) obtained a full equivalence with deterministic refuters for functions computable in P against probabilistic algorithms that go through a compression phase.

In the AM setting, Van Melkebeek and Sdroievski (CCC'23) established that targeted hitting-set generators are near-equivalent with lower bounds for multi-bit functions against algorithms on almost-all inputs. In this paper, we establish a full equivalence with nondeterministic refuters for functions computable in NP against Arthur-Merlin protocols that go through a compression phase. We also investigate applications of targeted hitting-set generators in the AM setting to explicit constructions, and argue that they imply regular hitting-set generators for MA.

1 Introduction

A central problem in the theory of computing is whether bounded-error probabilistic algorithms can be simulated deterministically with a polynomial overhead in time (the BPP versus P problem). In the context of interactive proofs, the corresponding problem is whether constant-round interactive proofs, commonly referred to as Arthur-Merlin protocols, can be simulated *nondeterministically* with a polynomial overhead in time (the AM versus NP problem). Classical hardness vs. randomness tradeoffs conclude derandomization from lower bounds, and in some cases equivalences are known. As the lower bound hypotheses are conjectured to hold, so are the conclusions BPP = P and AM = NP. However, in both settings even subexponential derandomizations remain open.

BPP setting. Linear-exponential circuit lower bounds for $\mathrm{E} \doteq \mathrm{DTIME}[2^{O(n)}]$ imply the existence of pseudo-random generators (PRGs) that achieve the derandomization prBPP $\subseteq$ P, where prBPP denotes the promise problem version of BPP [NW94, IW97]. Weaker circuit lower bounds yield PRGs achieving weaker derandomization results. At the low end of the derandomization spectrum, polynomial circuit lower bounds for EXP result in subexponential-time simulations for prBPP. PRGs lead to *blackbox* derandomization as they produce a small set of random-bit strings on which the acceptance probability of the randomized process under consideration is approximately the same as on all random-bit strings, for any input of a given length. In fact, blackbox derandomization is equivalent to circuit lower bounds, and the equivalence holds over the entire derandomization spectrum [SU05, Uma03].

It is open whether blackbox derandomization is as powerful as general, so-called *whitebox*, derandomization. Goldreich [Gol11] argued that whitebox derandomization of prBPP is equivalent to seemingly weaker objects than PRGs, namely *targeted* PRGs, which take an input x for the underlying randomized process and produce a small set of random-bit strings on which the acceptance probability approximates the true acceptance probability on that specific input x. Chen and Tell [CT21] raised the question of an equivalent lower bound and presented a near-equivalence in terms of uniform lower bounds for multi-bit functions that hold on almost-all inputs, in the sense that every algorithm in the class for which the lower bound holds can only compute the hard function on finitely many inputs. Their result falls short of a full-fledged equivalence due to an additional depth restriction in the direction from hardness to derandomization.

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]. For future reference, we state their main result in our notation (explained after the statement).[1]

[1]Chen, Tell and Williams [CTW23] state their main result in terms of refuters against efficient probabilistic *streaming* algorithms that run in small space. We believe, however, that our TICOMP notation better captures the

Theorem 1 ([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* $\mathrm{prBPTICOMP}[n^{1+\epsilon}, n^{\epsilon}]$.
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* $\mathrm{prBPTICOMP}[n^{a+\epsilon}, n^{\epsilon}]$.
3. $\mathrm{prBPP} \subseteq \mathrm{P}$.

For a class $\mathcal{A}$ of algorithms, $\mathcal{A}\mathrm{TICOMP}[t(n), s(n)]$ denotes the class of computational processes obtained by first running a probabilistic algorithm A_{comp} and then an algorithm $A_{\mathrm{dec}} \in \mathcal{A}$ 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} is able to compute the output. We refer to a pair $(A_{\mathrm{comp}}, A_{\mathrm{dec}})$ as a *bottleneck* algorithm. 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 item 2 in Theorem 1 is a relaxation of item 1. Note also that the existence of the refuter in item 1 or 2 only guarantees that, for any fixed $A' = (A_{\mathrm{comp}}, A_{\mathrm{dec}})$ in $\mathrm{prBPTICOMP}[t(n)^{1+\epsilon}, n^{\epsilon}]$, there exist infinitely many inputs on which A' fails to compute the function under consideration. This stands in contrast with the setting of hardness on almost-all inputs, where A' can only succeed on finitely many inputs. However, in the refutation setting the counterexamples need to be found efficiently.

AM setting. An equivalence between circuit lower bounds and blackbox derandomization is known throughout the entire spectrum [KvM02, MV05, SU05]. The role of EXP is now taken over by NEXP ∩ 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 AM setting 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.

Until recently, not much was known in the *whitebox* setting for AM. In [vMMS23a] we established a near-equivalence with hardness on almost-all inputs. In one direction, we showed that if there is a length-preserving function f computable in nondeterministic polynomial time that is hard on almost-all inputs against faster promise Arthur-Merlin protocols, then $\mathrm{prAM} \subseteq \mathrm{NP}$. In the other direction, assuming $\mathrm{prAM} \subseteq \mathrm{NP}$, we observed that there exists a length-preserving function f computable in nondeterministic polynomial-time with a few bits of advice that is hard against Arthur-Merlin protocols. Whereas in the corresponding results in the BPP setting, the remaining gap is an additional depth restriction in the first direction, here there is the advice required in the second direction and the distinction between regular and promise Arthur-Merlin protocols.

essential characteristics of the notion of efficient compression that is equivalent to derandomization.

A more significant difference between the two settings is that, in contrast to prBPP, for prAM it remains open whether whitebox derandomization is equivalent to targeted hitting-set generators — a question originally raised by Goldreich in [Gol11]. As a byproduct to our prior results [vMMS23a], we presented a first step toward a positive resolution: Any low-end derandomization of prAM implies the existence of a targeted hitting-set generator that achieves a slightly weaker derandomization. We also showed that targeted hitting-set generators are necessary for *mild* derandomization of prAM, i.e., simulations on NP-oracle or Σ_2 machines [vMMS23b]. Along the way, we obtained an equivalence between mild derandomization of prAM and hardness in the presence of efficiently-computable leakage.

Our results. As our main result, we establish a full equivalence between derandomization of Arthur-Merlin protocols via targeted hitting-set generators and refutation. The role of a bottleneck algorithm $(A_{\text{comp}}, A_{\text{dec}})$ in the BPP setting is taken over by a *bottleneck protocol* $(A_{\text{comp}}, P_{\text{dec}})$, which consists of a probabilistic compressor A_{comp} followed by an Arthur-Merlin protocol P_{dec} on the compressed input. More precisely, 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. Here, soundness of $(A_{\text{comp}}, P_{\text{dec}})$ for a function f means that for all inputs z, with high probability, $P_{\text{dec}}(A_{\text{comp}}(z))$ either correctly computes $f(z)$ or else indicates failure.

Theorem 2 (Main result). *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* $\text{prAMTICOMP}[n^{1+\epsilon}, n^{\epsilon}]$ *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* $\text{prAMTICOMP}[n^{a+\epsilon}, n^{\epsilon}]$ *protocols with promised soundness for f.*

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

Consider item 1 in Theorem 2. 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 2 shows that for derandomizing prAM, it suffices to derandomize trivial refuters for the identity function. In fact, as the relaxation in item 2 states, it suffices to derandomize trivial refuters for any function computable in NP.

Theorem 2 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 $\text{poly}(T(\text{poly}(n)))$. Similarly, a targeted hitting-set generator that runs in time T, and thus achieves the derandomization $\text{prAM} \subseteq \text{NTIME}[T(\text{poly}(n))]$, 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.

As mentioned before, in [vMMS23b] we showed that *mild* derandomization for prAM implies the existence of targeted hitting-set generators achieving the same derandomization result. Taken together with Theorem 2, we obtain an equivalence between mild derandomization and refutation. As with the equivalence of Theorem 2, the equivalence of Corollary 3 scales smoothly in terms of the running time for the refuter.

Corollary 3. *Let $\mathcal{C} \in \{\mathrm{P}^{\mathrm{NP}}, \mathrm{ZPP}^{\mathrm{NP}}, \Sigma_2\mathrm{P}\}$. The following are equivalent:*

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

3. *There exists a targeted hitting-set generator for* prAM *computable in $\mathcal{C}$.*

4. $\mathrm{prAM} \subseteq \mathcal{C}$*.*

The targeted HSGs in Theorem 2 are *multi-valued* nondeterministic algorithms in the sense that they produce a possibly different targeted hitting-set on each accepting computation path (and have at least one accepting computation path). Earlier papers on HSGs for prAM considered single-valued constructions. If we restrict the function f to be computable in deterministic polynomial time, the equivalence of Theorem 2 extends to targeted hitting-set generators for prAM that are deterministic, single-valued or deterministic with (non-adaptive) NP oracle algorithms provided the refuting algorithm is of the same type.

Multi-valued HSGs for prAM are sufficient for nondeterministic whitebox simulations and are arguably more natural than single-valued ones. Among other things, for the multi-valued versions, targeted HSGs for prAM imply regular HSGs for prMA. The result scales smoothly and can be stated in terms of nondeterministic PRGs for prBPP.

Theorem 4. *If there exists a targeted hitting-set generator for* prAM *computable in nondeterministic time $T(m)$, then there exists a pseudo-random generator for* prBPP *computable in nondeterministic time* $\mathrm{poly}(T(\mathrm{poly}(m)))$*.*

Assuming a positive resolution to Goldreich's question, Theorem 4 states that whitebox derandomization of prAM implies blackbox derandomization of prMA. A related result is that low-end whitebox derandomization of prMA implies the same derandomization in a blackbox fashion [IKW02]. A similar implication holds for mild derandomization of prAM [AvM17]. It is also known that whitebox derandomization of prAM implies a P^{NP} blackbox simulation of the same strength for prMA [AGHK11].

Building HSGs and PRGs are instantiations of a more general problem of *explicit constructions*. Many objects of interest — including HSGs and PRGs — can be shown to exist using the probabilistic method. Typically, this yields an efficient randomized algorithm that, on input 1^n, outputs an object of size n that has the desired property Π with high probability. One approach to obtain an explicit object with property Π is to run a targeted HSG that fools Π, cycle through the outputs, and either use an algorithm for Π to select an output that has property Π or else combine the outputs into a single (somewhat larger) object with property Π. For several objects of interest, the underlying Π is known to be in P, in which case a targeted HSG for prBPP suffices. For several more, including truth-tables of high circuit complexity and rigid matrices, Π is known to be in coNP, in which case a targeted HSG for prAM suffices. Depending on the complexity of the targeted HSG, the resulting explicit construction may be deterministic, nondeterministic single-valued, non-deterministic multi-valued, deterministic with an NP oracle, etc. We state the approach in the AM setting assuming nondeterministic multi-valued targeted HSGs. We use the term *probabilistic construction* for an efficient incarnation of the probabilistic method, i.e., a polynomial-time randomized algorithm that, on input 1^n, outputs a string $x \in \Pi$ of length at least n with probability at least $1/2$.

Proposition 5. *Assume there exists a targeted hitting-set generator for* prAM *computable in nondeterministic time T and let Π be a property that respects the following conditions:*

1. *Π is decidable in* coNP *and admits a probabilistic construction.*
2. *There exists a polynomial-time algorithm that, given a list $x_1, \ldots, x_k$ of strings in $\{0,1\}^n$ containing at least one $x_i \in \Pi$, outputs a string in Π of length at least n.*

Then there exists a nondeterministic algorithm that has an accepting computation path on every input and, on input 1^n, runs in time $\mathrm{poly}(T(\mathrm{poly}(n)))$ and outputs on every accepting computation path a string in Π of length at least n.

Explicit constructions for properties Π satisfying the conditions of Proposition 5 were obtained under non-uniform hardness conditions for E in [KvM02, SU06] and under a slight variation of the uniform hardness assumption $\mathrm{E} \not\subseteq$ io-AMTIME$[2^{\epsilon n}]$ for some $\epsilon > 0$ in [GSTS03]. By Theorem 2 and its extensions, we can obtain such explicit constructions under the refutation hypothesis of Theorem 2.

Techniques. To establish the direction from refutation to derandomization of our main result, we refine the instance-wise transformation of hardness into targeted hitting sets from our earlier work [vMMS23a]. The approach in [vMMS23a] extracts hardness from a nondeterministic computation on a given input z based on probabilistically checkable proofs, and employs the recursive Miltersen-Vinodchandran hardness-based HSG construction (RMV) due to Shaltiel and Umans [SU09]. In order to obtain the desired compression during reconstruction, we switch from PCPs to PCPs of proximity, among other changes.

In more detail, consider a function f computable in nondeterministic polynomial time that is hard on almost-all inputs against faster promise Arthur-Merlin protocols. To derandomize a prAM protocol P on input x, the approach in [vMMS23a] uses the PCP witnesses asserting the value of $f(x)$ as a basis for the RMV generator. In case the resulting output fails to be a hitting-set for P on input x, the RMV reconstructor with input $D \doteq P(x, \cdot)$ allows for compression of the PCP, which ultimately leads to a fast promise Arthur-Merlin protocol P_{rec} for computing f on any input z. In the refutation setting, we employ a refuter to obtain an input z such that P_{rec} instantiated with P and x fails to compute $f(z)$. This way, we can ensure that the hitting-set on input x succeeds. The refuter is only guaranteed to work against bottleneck protocols, though, and the reconstructor from [vMMS23a] does not offer the required compression due to the fact that the PCP verifiers need full access to the input z. A first idea is to, in addition to the PCP witnesses, use the input z itself as a basis for the RMV generator. In case the resulting generator fails, it is then possible to compress z using the RMV reconstructor. This allows us to obtain a bottleneck protocol by first compressing z and then feeding the compressed representation of z into P_{rec}, which computes

$f(z)$ from the compressed representation using the PCP strategy and the RMV evaluator to answer queries to z.

This gives us almost what we need. Due to the need for P_{rec} to run the polynomial-time PCP verifier, there is a multiplicative polynomial-time overhead (in $|z|$) for the reconstructor, which is too inefficient. In order to improve the efficiency, we instead employ PCPs of proximity (PCPPs) together with an error-correctable encoding of the input z. As there exist PCPPs that run in subpolynomial time and only require oracle access to the input, the overhead from running them becomes sublinear in the time for the remaining steps of the reconstructor.

For the other direction of the equivalence — that targeted generators sufficient for derandomizing prAM imply refutation — we employ such generators to derandomize the process of obtaining a counterexample at random. Straightforwardly, this would require a targeted generator that fools P^{prAM} rather than prAM because we know how to verify the validity of a counterexample in the former but not in the latter class. The hypothesis only gives us a targeted generator that fools prAM, though. To bridge the gap, we make sure our reconstructor possesses a strong form of soundness, which we dubbed *resilient soundness* and established for our regular reconstructor in [vMMS23a]. The resilient soundness property allows us to focus on refuting bottleneck protocols with promised soundness. In this case, verifying whether a counterexample is valid becomes a prAM computation, so we can get by with a targeted generator that fools prAM for the derandomization. We similarly employ targeted generators to obtain explicit constructions from the probabilistic method (Proposition 5), including truth-tables with high circuit complexity.

Organization. In Section 2, we develop the ideas behind our main result and relate them to existing techniques. We start the formal treatment in Section 3 with definitions, notation, and other preliminaries. In Section 4, we present the details of our main result. In Section 5, we discuss explicit constructions: Proposition 5 and examples of conditional explicit constructions, including hard truth-tables, rigid matrices, and Theorem 4.

2 Technical overview

We start with a recap of the techniques involved in the hardness vs. randomness tradeoffs for BPP leading up to Theorem 2.

BPP setting. The known hardness vs. randomness tradeoffs are based on a pseudo-random generator construction G that takes a function h and outputs a pseudo-random distribution G^h. These generators are typically *learning*, meaning that any statistical test D that distinguishes G^h from uniform suffices as an oracle to efficiently learn h from a small number of queries. In case G^h does not "fool" an efficient randomized algorithm A on input x, the function h can be learned efficiently from the distinguisher $D(r) \doteq A(x, r)$ and a small number of evaluations of h, where $A(x, r)$ the output of A on input x and random-bit string r. Given a learning PRG construction G, one can construct a *targeted* PRG by instantiating G with an oracle $h = h_x$ that depends on x. The question is how to construct h_x out of x. We now survey the known constructions.

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]. Their reconstructor is based on a bootstrapping strategy similar to [IW01] that uses the NW reconstructor to obtain, layer-by-layer, small circuits encoding the gate values for the circuit computing $f(x)$. Because the bootstrapping strategy requires the NW reconstructor to work for all layers, Chen and Tell only end up with a (targeted) *hitting-set* generator rather than a *pseudo-random* generator.

The Chen-Tell approach hinges on the speed of the reconstruction process. Subsequent works exploit 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.

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.

Each of the above approaches can be viewed as an explicit construction of one or more h_x from x such that

$$A_{\text{rec}}(h_x, D) \neq h_x \tag{1}$$

for at least one h_x, where A_{rec} denotes the output of the reconstructor (which only needs access to D and the answers to the learning queries to h_x). Such an explicit construction suffices because (1) 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 (1) 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 (1) 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.

AM setting. Some changes are needed when making the transition to the AM setting. 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. Moreover, since the objective is to obtain a nondeterministic simulation for prAM, the algorithms for computing f, the refuters and the targeted generators are all allowed to be nondeterministic

In [vMMS23a], we built a targeted hitting-set generator for AM based on the recursive Miltersen-Vinodchandran hitting-set generator due to Shaltiel and Umans [SU09]. To obtain h_x from x in the setting of hardness on almost-all inputs, we make use of 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 $\text{polylog}(T(n))$ 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. To guarantee soundness, we rely on a *resilience* property of the RMV generator, which was first observed in [GSTS03] for the Miltersen-Vinodchandran generator [MV05]. The resilience property 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 6. *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.*

In the setting of hardness on almost-all inputs, the co-nondeterministic circuit D is obtained as $P(x, \cdot)$, where x is the input for which we want to derandomize an Arthur-Merlin protocol P. This is, however, not essential for the construction.

From refutation to targeted-generators. 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 6 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 6 does not hold). In that case, $H(z, D)$ must output a hitting set for D (the first item in Property 6 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 show that item 2 in Theorem 2 implies item 3.

Targeted generators to refutation. For the implication from item 3 to item 1 in Theorem 2, assuming the existence of a targeted HSG sufficient for derandomizing prAM, we need to exhibit a refuter for identity against polynomial-time bottleneck Arthur-Merlin protocols with subpolynomial compression bottlenecks. Fix such a 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, so we only need a refuter against bottleneck protocols that are sound. 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.

3 Preliminaries

We assume familiarity with standard complexity classes such as NP, AM, and prAM. As is customary, all time bounds t are implicitly assumed to be time-constructible and satisfy $t(n) \geq n$.

3.1 Nondeterministic and co-nondeterministic computation

We make use of nondeterministic and co-nondeterministic 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.

We are also interested in nondeterministic algorithms that compute total relations $R \subseteq \{0,1\}^* \times \{0,1\}^*$. 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 on all successful computation paths, $N(x)$ outputs some y such that $R(x, y)$ holds, where y can depend on the computation path. 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.

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 7 (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: $(\forall M) \Pr[P(x, M) \notin \{v, \perp\}] \leq s$.*

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 $s(|x|)$.

3.3 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 8. *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 8 applies to promise Arthur-Merlin protocols that output values, yielding the bottleneck protocol classes prAMTICOMP$[t(n), s(n)]$. In the completeness and soundness notions of Definition 7, 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).

3.4 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 9. *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 9 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.

3.5 Hitting-set generators and targeted hitting-set generators

Targeted generators typically either receive an input x and produce a set that (for sufficiently large $|x|$) is pseudo-random for every algorithm A with input x; or receive as input a circuit D (usually taken to be $A(x, \cdot)$) and produce a set that is pseudorandom for D. Both definitions are equivalent [Gol20], and we might as well give the targeted generator access to both x and a circuit C representing A such that $D = C(x, \cdot)$ [vMMS23a]. 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.

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

Definition 10 (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 11 (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.

3.6 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 12 (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 13 ([BGH$^+$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 $d(\mathrm{Enc}(z), \mathrm{Enc}(z')) \geq \delta$. A decoder Dec for an ECC with distance parameter δ is an algorithm that, on input a possibly corrupted codeword $\tilde{z}$ in the co-domain of Enc, recovers z as long as $d(\tilde{z}, \mathrm{Enc}(z)) < \delta/2$. For our purposes, it suffices that for any constant $\delta \in (0,1]$ there exists an ECC with distance parameter δ that is computable and decodable 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.

4 Equivalence result

In this section, we establish Theorem 2. First, in Section 4.1, we present the refinement for the targeted hitting-set generator of [vMMS23a]. In Section 4.2, we prove the direction of refutation to targeted hitting-set generators. In Section 4.3, we prove the direction of targeted hitting-set generators to refutation. Finally, we put everything together to obtain our equivalence result and extensions in Section 4.4

4.1 Targeted generator construction

We develop our targeted HSG as a refinement of our earlier work. In this section, we describe the modifications needed to the construction and proof in [vMMS23a, Theorem 24]. For completeness and for readers not familiar with our earlier work, we include a full proof in the appendix.

Theorem 14. *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 (sketch). The proof is similar to [vMMS23a, Theorem 24], with the following differences:

- We use PCPPs in place of PCPs.
- The generator additionally instantiates the RMV generator with an encoding of the input z.
- We strengthen the reconstructor to a bottleneck protocol.

Consider the language L_f that consists of strings $(\tilde{z}, n, i, b)$ such that $\tilde{z} = \mathrm{Enc}(z)$ for some $z \in \{0,1\}^n$ and $f(z)_i = b$, where Enc is a suitable error-correcting code as in Section 3.6. Note, in particular, that L_f is computable in nondeterministic time $n \cdot \mathrm{polylog}(n) + T(n)$. Let V be the PCPP verifier for L_f given by Lemma 13, where we consider $\tilde{z}$ as an implicit input and the remaining part of the input as explicit.

The generator H, on input z and a co-nondeterministic circuit D of length m, computes $\tilde{z} = \mathrm{Enc}(z)$, guesses $v = f(z)$ and, for every i, guesses and verifies a PCPP y_i that asserts the i-th bit of $f(z)$. H then computes suitable low-degree extensions $\hat{z}$ for $\tilde{z}$ and $\hat{y}_i$ for each y_i, and outputs $\mathrm{RMV}(\hat{z}) \cup \bigcup_{i \in [|v|]} \mathrm{RMV}(\hat{y}_i)$. The running time for the generator is dominated by the running time for running the RMV generator.

The reconstructor has two parts, the compressor A_{comp} and the decompressor P_{dec}. A_{comp}, on input $z \in \{0,1\}^n$ and 1^m, computes $\tilde{z}$ and a commitment/sketch π_z for the RMV reconstructor for $\tilde{z}$. Due to the inherent compression of the RMV reconstructor, π_z has length $\mathrm{poly}(m, \log T(n))$. We now describe P_{dec} on input π_z, a co-nondeterministic circuit D of size m and an index i. First, the honest Merlin sends a bit b and commits to the low-degree extension of a PCPP y_i asserting that $f(z) = i$. Since Merlin may be dishonest, let $\tilde{y}_i$ denote the value Merlin committed to. Arthur then computes $V^{\tilde{z}, \tilde{y}_i}(n, i, b)$, employing Merlin's help and the RMV evaluator to answer queries to $\tilde{z}$ and $\tilde{y}_i$. Finally, Arthur either succeeds and outputs b, if the verification using V is successful, or fails otherwise. The resilience property for the RMV reconstructor guarantees that, with high probability, every execution of the RMV evaluator leads to an evaluation of a fixed string $\tilde{y}$. Then, soundness for $(A_{\mathrm{comp}}, P_{\mathrm{dec}})$ follows from the soundness of V itself. As for completeness, in case $H(z, D)$ fails to hit D, then the RMV construction instantiated with D allows for compressing both $\tilde{z}$ and each y_i, guaranteeing completeness for our construction. The PCPP verifier is very efficient, running in time $\mathrm{poly}(m, \log T(n))$. For this reason, the running time for our reconstructor is dominated by the running time for the RMV reconstructor. □

We remark that if the function f in Theorem 14 is computable in deterministic time T, then we can modify the construction so that the generator H runs in *deterministic* time $\mathrm{poly}(T(n), m)$. To do so, we similarly define L_f as the set of $(\tilde{z}, n, i, b)$ such that $\mathrm{Dec}(\tilde{z}) = z$ for some $z \in \{0,1\}^n$ and $f(z)_i = b$, and employ a canonical PCPP construction as in [Par21] interpreting (n, i, b) as the explicit input and $\tilde{z}$ as the implicit input. With these changes, it follows that $L_f \in \mathrm{DTIME}[n \cdot \mathrm{polylog}(n) + T(n)]$, and employing a canonical PCPP construction implies that the generator H can deterministically compute, for each i, the only PCPP y_i that is accepted with probability 1 by the PCPP verifier, ultimately obtaining a deterministic H.

4.2 From refutation to derandomization

We now show that the second item in Theorem 2 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 14 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 14 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 14 fails for z_i). It follows that $H(z_i, D)$ hits D (item 1 in Theorem 14 holds).

Theorem 15. *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 14 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 $P_{\mathrm{dec}}(\cdot, D)$ be the protocol P_{dec} with the circuit D fixed as its second input. The generator then feeds inputs 1^n and $(A_{\mathrm{comp}}(\cdot, 1^m), P_{\mathrm{dec}}(\cdot, D))$ 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 14 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 14 guarantees that $(A_{\mathrm{comp}}(\cdot, 1^m), P_{\mathrm{dec}}(\cdot, D))$ is sound for f. To ensure correctness of the generator, we set the value of n sufficiently large such that $A_{\mathrm{comp}}(\cdot, 1^m)$ and $P_{\mathrm{dec}}(\cdot, D)$ run in time at most $n^{a+\epsilon}$ and such that the output length of $A_{\mathrm{comp}}(\cdot, 1^m)$ 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 $(A_{\mathrm{comp}}(\cdot, 1^m), P_{\mathrm{dec}}(\cdot, D))$ fails to compute $f(z_i)$ with completeness 1 and soundness 1/3. This means that item 2 in Theorem 14 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 $z \in \{0,1\}^n$, $A_{\mathrm{comp}}(\cdot, 1^m)$ outputs a string of length $\mathrm{poly}(m, a\log n) \le (m \cdot \log n)^{k_1}$ for a fixed constant k_1. Moreover, the running time for $A_{\mathrm{comp}}(\cdot, 1^m)$ is $n \cdot \mathrm{poly}(m, a\log n) = n \cdot \mathrm{poly}(m, \log n) \le n \cdot (m \cdot \log n)^{k_2}$ for some constant k_2, and the running time for $P_{\mathrm{dec}}(\cdot, D)$ 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 $A_{\mathrm{comp}}(\cdot, 1^m)$ 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 $A_{\mathrm{comp}}(\cdot, 1^m)$ 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 $P_{\mathrm{dec}}(\cdot, D)$ 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 $(A_{\mathrm{comp}}, P_{\mathrm{dec}})$. It follows that $(A_{\mathrm{comp}}(\cdot, 1^m), P_{\mathrm{dec}}(\cdot, D))$ 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 $T(n + m') = T(\mathrm{poly}(m))$, 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 $\mathrm{poly}(T(\mathrm{poly}(m)))$, which dominates the running time for the generator. □

In contrast to the corresponding result of [CTW23] in the BPP setting, Theorem 15 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. The time T arguably is the most natural choice of scaling parameter since it translates directly into slower hitting-set generators without any extra steps such as increasing the input length. One can also consider scaling with respect to secondary parameters, namely 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 behavior of the RMV reconstructor at the low end, our approach does not reach the low end when scaling those secondary parameters. 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 16. *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 $2^{\text{polylog}(n)}$, then there is a targeted hitting-set generator for* prAM *that is computable in nondeterministic time $2^{\text{polylog}(n)}$.*

4.3 From derandomization to refutation

Next, we prove that the third item in Theorem 2 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 17). The first item then follows as the identity function has such a probabilistic refuter (Proposition 18).

A probabilistic refuter is a refuter that produces a counterexample with constant probability over its internal randomness. In the case of the class $\text{prAMTICOMP}[t(n), s(n)]$ 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 $(A_{\text{comp}}, P_{\text{dec}})$ 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 $(A_{\text{comp}}, P_{\text{dec}})$ run in time $t(\ell)$ and the output length of A_{comp} is bounded by $s(\ell)$, then $(A_{\text{comp}}, P_{\text{dec}})$ fails to compute f on input z with completeness $2/3$ and soundness $1/3$. Note that if $(A_{\text{comp}}, P_{\text{dec}})$ is promised to be sound for f and to obey the time and compression requirements, then it must be the case that $(A_{\text{comp}}, P_{\text{dec}})$ fails to compute $f(z)$ with completeness $2/3$.

Here is the intuition for the stronger statement (Theorem 17). 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 $(A_{\text{comp}}, P_{\text{dec}})$ 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 $A_{\text{comp}}(z, r_2)$ 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 3.4. 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 14 has perfect completeness. The resilient soundness property of the reconstructor in Theorem 14 ensures that we only need to worry about refuting pairs $(A_{\text{comp}}, P_{\text{dec}})$ that are sound for f.

Theorem 17. *Assume 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 17 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 17. 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 $n \le \ell = O(\text{poly}(n))$. 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 δ 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' composed of k copies of D, which accepts if and only if any of the copies accept, for a constant k to be defined next. 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 $\text{prAMTICOMP}[t(n), s(n)]$ 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 18. *For every constant $\epsilon \in (0, 1)$, there exists a probabilistic polynomial-time refuter for the identity function against* $\text{prAMTICOMP}[\infty, n^\epsilon]$ *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. $\square$

4.4 Proof of equivalence result

We now have all the steps involved in Theorem 2; we just need to tie them together.

Proof of Theorem 2. The implication $1 \implies 2$ holds trivially by taking the identity for f. The implication $2 \implies 3$ is Theorem 15. The implication $3 \implies 1$ follows by combining Theorem 17 and Proposition 18 with polynomial time bounds. $\square$

The strategy also applies to other types of targeted HSGs such as deterministic, single-valued and deterministic with (parallel) NP oracle. Finally, we present a general version of Theorem 2 that applies to different types of targeted hitting-set generators for prAM.

Theorem 19. *Let $\mathcal{C} \in \{\mathrm{P}, (\mathrm{NP} \cap \mathrm{coNP}), \mathrm{P}^{\mathrm{NP}}_{||}, \mathrm{P}^{\mathrm{NP}}\}$. The following are equivalent:*

1. *For some constant $\epsilon \in (0,1)$, there exists a list-refuter computable in $\mathcal{C}$ 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* deterministic *time n^a that admits a list-refuter computable in $\mathcal{C}$ against* prAMTICOMP$[n^{a+\epsilon}, n^\epsilon]$ *protocols with promised soundness for f.*

3. *There exists a targeted hitting-set generator for* prAM *computable in $\mathcal{C}$.*

Proof (sketch). The equivalence goes along the same lines as that of Theorem 2, using the observation after Theorem 14 that if the algorithm computing f is deterministic, then the targeted HSG construction of Theorem 14 is also deterministic. In this case, assuming a refuter computable in $\mathcal{C}$ and following Theorem 15, we obtain a targeted HSG that is also computable in $\mathcal{C}$. Similarly and following Theorem 17, a targeted HSG computable in $\mathcal{C}$ leads to a refuter computable in $\mathcal{C}$. $\square$

The equivalence of Theorem 19 scales in the same way as that of Theorem 2, and in particular holds for quasipolynomial and subexponential time bounds. For completeness, we include an argument for Corollary 3.

Proof of Corollary 3. That (1) implies (2) is trivial. We show that (2) implies (3) by retracing the argument of Theorem 15. The original argument assumes that the refuter is computable in nondeterministic time T and concludes the existence of a targeted hitting-set generator that runs in nondeterministic time $\mathrm{poly}(T(\mathrm{poly}(n)))$. In the original argument, to obtain the targeted HSG, we run the refuter to obtain a list of inputs of length $n = \mathrm{poly}(m)$ (where m is the size of the co-nondeterministic circuit given as input to the targeted generator) and then use each input as a basis for the (nondeterministic) generator H of Theorem 14. For deterministic and zero-error probabilistic algorithms with oracle access to SAT, we can use the SAT oracle to compute the lexicographically least output of H after obtaining the output of the refuter. For a refuter computable by a Σ_2 algorithm, we can have the targeted generator guess the output of the refuter and of H, and then verify with an existential and a universal guess that the outputs of the refuter and H were guessed correctly. In all cases, the final running time for the targeted HSG is $\mathrm{poly}(T(\mathrm{poly}(m)))$, which is polynomial if T is polynomial.

To see that (3) implies (1), we retrace the proof of Theorem 17. In the proof, we construct a co-nondeterministic circuit D that verifies whether a given input serves as a counterexample for the input bottleneck protocol, and then run the assumed targeted hitting-set generator to produce a list of inputs containing a counterexample. On input 1^n and a bottleneck protocol of description length m, setting up D can be done deterministically in time $\mathrm{poly}(m, n)$. Thus, if we start from a targeted hitting-set generator computable in $\mathcal{C}$, we obtain a list-refuter computable in $\mathcal{C}$.

Finally, the equivalence of (3) and (4) follows from Theorem 5 in [vMMS23b]. □

5 Explicit constructions

In this section, we establish a connection between targeted generators for prAM and explicit constructions. We first establish our general result in Section 5.1 and then highlight some applications in Section 5.2.

5.1 General statement

We prove a more general version of Proposition 5 that applies for different types of targeted generators for prAM. For the upcoming statement, recall that a probabilistic construction for a property Π is a polynomial-time randomized algorithm that, on input 1^n, outputs a string $x \in \Pi$ of length at least n with probability at least $1/2$.

Proposition 20. *Let T be a time bound and $\mathcal{C} \in \{\mathrm{P}, (\mathrm{NP} \cap \mathrm{coNP}), \mathrm{NP}, \mathrm{P}^{\mathrm{NP}}_{||}, \mathrm{P}^{\mathrm{NP}}\}$. Assume there exists a targeted hitting-set generator for* prAM *of type $\mathcal{C}$ that is computable in time T and let Π be a property that respects the following conditions:*

1. *Π is decidable in* coNP *and admits a probabilistic construction.*
2. *There exists a polynomial time algorithm of type $\mathcal{C}$ that given a list $x_1, \ldots, x_k$ of strings in $\{0,1\}^n$ containing some $x_i \in \Pi$ outputs a string in Π of length at least n.*

Then there exists an algorithm of type $\mathcal{C}$ that, on input 1^n, runs in time $\mathrm{poly}(T(\mathrm{poly}(n)))$ and outputs a string in Π of length at least n.

Proof. Let A' be a probabilistic construction for Π, B the algorithm in the second item and H the assumed targeted generator for prAM. We describe an explicit construction A for Π: On input 1^n, A first constructs a co-nondeterministic circuit D that, on input a random sequence r for A', computes $v \doteq A'(1^n, r)$ and co-nondeterministically verifies that $v \in \Pi$. Then, A computes $H(D)$, obtaining a list of strings $r_1, \ldots, r_k$. Finally, A runs algorithm B on inputs $A'(1^n, r_1), \ldots, A'(1^n, r_k)$ and outputs whatever B does.

To see that A is correct, we note that because Π has a probabilistic construction and $\Pi \in \mathrm{coNP}$, it follows that D accepts at least a half of its inputs, while only accepting strings r such that $A'(1^n, r) \in \Pi$. In that case, the generator H on input D is guaranteed to output a list $r_1, \ldots, r_k$ such that there exists r_i for which $A'(1^n, r_i) \in \Pi$. Then, the guarantee on B implies that A outputs a string in Π of length at least n. The circuit D has size $m = \mathrm{poly}(n)$, and it can be constructed in that time. Then, computing $H(D)$ takes time $T(\mathrm{poly}(n))$, and thus produces a list of strings of size at most $T(\mathrm{poly}(n))$. Finally, running B on the resulting list takes time $\mathrm{poly}(T(\mathrm{poly}(n)))$.

For the case $\mathcal{C} = \mathrm{P}_{||}^{\mathrm{NP}}$, the construction above has two rounds of non-adaptive NP queries. Using standard results (see, e.g., [SU06, Lemma 7.2]), we can obtain a construction with a single round of non-adaptive NP queries while incurring a polynomial time overhead. $\square$

5.2 Instantiations

In this section, we provide examples of explicit constructions that can be obtained by applying Proposition 20.

Nondeterministic construction of hard truth-tables. We show how to construct truth-tables of high circuit complexity assuming the existence of targeted generators sufficient to derandomize prAM. Then, using the construction, we prove Theorem 4.

Theorem 21. *Assume there exists a targeted hitting-set generator for* prAM *computable in nondeterministic time $T(m)$. Then, there exists a nondeterministic algorithm that always has an accepting computation path and, on input 1^m, runs in time $T(\mathrm{poly}(m))$ and outputs, on every accepting computation path, the truth-table of a function with circuit complexity at least m.*

Proof. Assume there is a targeted hitting-set generator for prAM that is computable in nondeterministic time T. It suffices to show that truth-tables of high circuit complexity respect the first and second items in Proposition 20. Testing whether a truth-table x has high circuit complexity can be done in coNP by universally guessing a circuit of size s and checking that the guessed circuit fails to compute x. We now describe a probabilistic construction A' for the property: On input 1^s, the algorithm A' outputs a random truth-table of a Boolean function on $2\lceil \log s \rceil$ input bits. Since there are at least 2^{s^2} possible such truth-tables but only $2^{O(s \log s)}$ circuits of size at most s, A' succeeds with probability at least $1/2$ for sufficiently large s. For the second item, we note that one can just concatenate (padding with zeroes in the end if necessary) a list $x_1, \ldots, x_k$ of truth tables that contains a x_i of circuit complexity at least s to obtain a single truth-table of circuit complexity at least s. $\square$

Theorem 4 follows from Theorem 21 and traditional hardness vs. randomness tradeoffs [Uma03]. The conclusion of Theorem 21 together with the construction in [Uma03] results in a nondeterministic PRG that, on input 1^m, runs in time $\mathrm{poly}(T(\mathrm{poly}(m))$ and fools every circuit of size m, as desired.

Deterministic construction of rigid matrices. The rank-r rigidity of a matrix M over a ring S, which we denote by $R_M^S(r)$, is the minimum number of entries of M that must be changed so that the rank of M becomes r or below. The maximum r-rigidity of an $n \times n$ matrix M is $(n-r)^2$, and in [Val77], Valiant showed that most matrices have very high rigidity $(n-r)^2/\log n$. However, known deterministic explicit constructions do not achieve this rigidity (see [Ram20] for some recent progress). We show that a deterministic construction of matrices with very high rigidity follows from the existence of deterministic refuters for a function f computable in deterministic polynomial-time against bottleneck protocols.

Theorem 22. *Assume that for some constants $a \geq 1$ and $\epsilon \in (0,1)$, there exists a function f computable in deterministic time n^a that admits a deterministic polynomial time list-refuter against* $\mathrm{prAMTICOMP}[n^{a+\epsilon}, n^{\epsilon}]$ *protocols with promised soundness for f. Then, for any prime p, there is a* *deterministic polynomial-time algorithm that, on input n, outputs a matrix M_n over the polynomial ring $\mathbb{F}_p[x]$ such that $R_{M_n}^{\mathbb{F}_p[x]} = \Omega((n-r)^2/\log n)$.*

Proof (sketch). We first use Theorem 19 to conclude the existence of a deterministic polynomial-time targeted hitting-set generator for prAM. We now argue that the rigidity property for matrices obeys the conditions of Proposition 20. Testing if a matrix is rigid is in coNP by universally guessing a matrix A with "few" entries and verifying that the rank of $M + A$ is larger than r. The result of Valiant [Val77] implies a probabilistic construction of rigid matrices over $\mathbb{F}_p$ for any prime p, and Klivans and van Melkebeek [KvM02] show how to obtain a single rigid matrix (though over the polynomial ring $\mathbb{F}_p[x]$) from a list of matrices that is guaranteed to contain a rigid matrix. $\square$

The reason we present the precondition of Theorem 22 in terms of the existence of refuters is to provide a (constructive) hardness condition under which constructing rigid matrices is possible, following other works that construct rigid matrices under similar assumptions [KvM02, GSTS03].

Appendix – Detailed targeted hitting-set generator construction

For completeness, we provide the full analysis of the targeted HSG that we need for our main result. In Appendix A, we describe the RMV generator and its reconstructor. In Appendix B, we prove Theorem 14.

A 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 23 (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 24 (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 24. 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.[2] Shaltiel and Umans call these MV lines, which we define next.

[2]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.

Definition 25 (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 24 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 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], but before presenting it we define commit-and-evaluate protocols.

A *commit-and-evaluate* protocol [SU09] has the syntactic structure of a pair of promise Arthur-Merlin protocol. The first protocol of the pair represents a commitment phase. In this phase, Arthur and Merlin interact and produce an output π, which we call a commitment. The commitment is given as input to the protocol of the evaluation phase. Even 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. For the upcoming definition, we say that a prAM protocol P has partial single-valuedness s if there exists a value v such that P outputs v with soundness s.

Definition 26 (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 (2) is at most $r(n)$:*

$$\text{For every } x \text{ of length } n \text{ in } X,\ P_{eval}(x, \pi) \text{ has partial single-valuedness } s(n). \tag{2}$$

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).

We now present the RMV reconstructor 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 27 ([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.*

- 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.

B Targeted generator and reconstruction

In this appendix, we present our targeted HSG construction in more detail, which works as follows: On input z and a co-nondeterministic circuit D of size m, it guesses a PCPP (as in Lemma 13) 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:

Proof of Theorem 14. 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 3.6. 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 $\tilde{z} = \text{Enc}(z)$ for $z \in \{0,1\}^n$ such that $f(z)_i = b$. In particular, L_f is decidable in nondeterministic time $n \cdot \text{polylog}(n) + T(n)$ by guessing $z \in \{0,1\}^n$, computing $\text{Enc}(z)$, checking that $\text{Enc}(z) = \tilde{z}$ and computing $f(z)_i$. Let V be the PCPP verifier given by Lemma 13 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 $\text{poly}(T(n), n) = \text{poly}(T(n))$ 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 $\text{poly}(T(n))$. 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 $h^r = \text{poly}(T(n), m)$ 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 $\tilde{z} = \text{Enc}(z)$ 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 $\text{poly}(T(n), m)$ 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 $\text{RMV}(\hat{z})$ and $\cup_{i \in [|v|]} \text{RMV}(\hat{y}_i)$, 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 27.

Computing $\tilde{z}$ and the initial verification step takes time $\text{poly}(T(n), m)$, computing the low-degree extensions for the PCPPs also takes time $\text{poly}(T(n), m)$ and each execution of the RMV generator, including the one for $\hat{z}$, takes time $p^{O(r)} = \text{poly}(T(n), m)$and outputs strings of length m. This culminates in a running time of $\text{poly}(T(n), m)$. 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 $(A_{\text{comp}}, P_{\text{dec}})$, which uses the commit-and-evaluate protocol $(P_{\text{commit}}, P_{\text{eval}})$ of Lemma 27 with fixed input p and resilience parameter $s = s(m,n) = (100q)^{-1}$, where $q = q(m,n) = \text{polylog}(T(n), m)$ 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 $\tilde{z} = \text{Enc}(z)$. 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 27. 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 $|S^2| = \log(1/s) \cdot \text{poly}(\Delta, r) = \text{poly}(m, \log T(n))$ points in $\mathbb{F}^{r'}$ together with evaluations of $\hat{z}$ on each point, each of which can be described with $\log p = \text{polylog}(m, \log T(n))$ bits, resulting in a total output length of $\text{poly}(m, \log T(n))$.

Completeness. If D is not hit by $H(z, D)$, then $\text{RMV}(\hat{z})$ 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 $\text{RMV}(\hat{y}_i)$ 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 27. 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 27 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 13 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, $P_{\text{dec}}(D, (A_{\text{comp}}(1^m, z)))$ either fails or outputs a bit of $f(z)$ with probability at least 2/3. In particular, if completeness also holds then $P_{\text{dec}}(D, (A_{\text{comp}}(1^m, z)))$ 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 27 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. $\square$

References

[AGHK11] Barış Aydınlıoğlu, Dan Gutfreund, John M. Hitchcock, and Akinori Kawachi. Derandomizing Arthur-Merlin games and approximate counting implies exponential-size lower bounds. *Computational Complexity*, 20(2):329–366, 2011.

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

[BGH$^+$05] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Short PCPs verifiable in polylogarithmic time. In *Conference on Computational Complexity (CCC)*, pages 120–134, 2005.

[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.

[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] Jørn 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)*, pages 35:1–35:17, 2022.

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

[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.

[Par21] Orr Paradise. Smooth and strong PCPs. *Computational Complexity*, 30(1):1, 2021.

[Ram20] C. Ramya. Recent progress on matrix rigidity – A survey. *CoRR*, abs/2009.09460, 2020.

[Spi96] Daniel 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.

[SU06] Ronen Shaltiel and Christopher Umans. Pseudorandomness for approximate counting and sampling. *Computational Complexity*, 15(4):298–341, 2006.

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

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

[Val77] Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In *Mathematical Foundations of Computer Science (MFCS)*, pages 162–176, 1977.

[vMMS23a] Dieter van Melkebeek and Nicollas Mocelin Sdroievski. Instance-wise hardness versus randomness tradeoffs for Arthur-Merlin protocols. In *Computational Complexity Conference (CCC)*, pages 17:1–17:36, 2023.

[vMMS23b] Dieter van Melkebeek and Nicollas Mocelin Sdroievski. Leakage resilience, targeted pseudorandom generators, and mild derandomization of Arthur-Merlin protocols. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 29:1–29:22, 2023.