Derandomization vs. Lower Bounds for Arthur-Merlin Protocols

By

Nicollas Mocelin Sdroievski

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2024

Date of final oral examination: 07/31/2024

The dissertation is approved by the following members of the Final Oral Committee:

Dieter van Melkebeek, Professor, Computer Sciences
Alberto Del Pia, Associate Professor, Industrial and Systems Engineering
Rishab Goyal, Assistant Professor, Computer Sciences
Christos Tzamos, Associate Professor, Informatics and Telecommunications at the University of Athens

Contents

Abstract

The subfield of derandomization in computational complexity theory aims to understand the conditions under which randomness can be efficiently reduced or completely eliminated from computational processes. The fundamental question regarding algorithms asks whether probabilistic algorithms can be simulated deterministically with a small overhead in time. A corresponding question about proofs asks whether probabilistic proofs known as Arthur-Merlin protocols can be simulated nondeterministically with a small overhead in time. Both questions are intricately tied to computational hardness. Prominently, in both settings *blackbox* derandomization, i.e., derandomization through pseudorandom generators, has been shown equivalent to lower bounds for decision problems against circuits and nondeterministic circuits, respectively. This dissertation focuses on generic, so-called *whitebox* derandomization for Arthur-Merlin protocols.

Instance-wise hardness. We develop a near-equivalence for Arthur-Merlin protocols between whitebox derandomization and lower bounds on almost-all inputs against protocols for multi-bit functions f that are efficiently computable nondeterministically. Our technique works instance-wise: It succeeds for every input x for which a particular Arthur-Merlin protocol fails to compute $f(x)$. We also obtain the first indication for Arthur-Merlin protocols of the equivalence between whitebox derandomization and the existence of targeted pseudorandom generators.

Refuting bottleneck protocols. We establish a full equivalence for Arthur-Merlin protocols between derandomization via targeted pseudorandom generators and the existence

of efficiently computable nondeterministic refuters for the identity function against *bottleneck* Arthur-Merlin protocols. We also show that the identity function can be replaced by any other function f that is efficiently computable nondeterministically. A refuter for a function f is a meta-algorithm that, on input the description of a protocol, finds an input where the protocol fails to compute f. An Arthur-Merlin protocol is said to have a bottleneck if it first probabilistically compresses the input to a shorter representation and then computes the output from the compressed representation.

Mild derandomization. We develop two equivalence results for *mild* derandomizations of Arthur-Merlin protocols: between whitebox derandomization and targeted pseudorandom generators, and between whitebox derandomization and the existence of multi-bit functions f that are efficiently computable nondeterministically and are leakage-resilient against Arthur-Merlin protocols. A mild derandomization is a simulation by a nondeterministic algorithm with oracle access to NP, as opposed to a simulation by a plain nondeterministic algorithm. A function f is leakage-resilient against a class of protocols if every protocol in the class fails to compute f even when additionally receiving a small amount of information about the value of f on the given input.

In addition to time-efficient derandomization of Arthur-Merlin protocols, we also study space-efficient derandomization of probabilistic algorithms. We show that space-efficient isolation for generic as well as catalytic computation follows from the existence of efficiently-computable unambiguous refuters against unambiguous space-bounded bottleneck algorithms. We also show that a space-bounded version of Arthur-Merlin protocols admit space-efficient catalytic algorithms. Isolation is the process of singling out solutions for computational problems that may have multiple solutions. The notion is closely related to unambiguous algorithms, i.e., nondeterministic algorithms that have at most one accepting computation path. Catalytic computation has read/write access to an additional populated memory that must be restored to its initial contents at the end of the execution.

Acknowledgments

This dissertation would not be possible without the help and support of several people throughout my studies. Please forgive me for any omissions.

First, I would like to thank my advisor, Dieter van Melkebeek. Thank you for guiding me not only from the beginning of the program, but from the moment you met me as I was still preparing to start my studies. Thank you for mentoring my research, teaching, and writing. Thank you for advocating for me, for your patience, and for the numerous meetings where you would help me develop initial ideas into actual research. I admire your dedication to teaching; you set an example I strive for.

To the committee members, Alberto Del Pia, Christos Tzamos, and Rishab Goyal: Thanks for the flexibility and all the work to get me through the end of the program. I would also like to thank Jin-Yi Cai for being part of my qualification exam, and all the instructors who helped me learn so much throughout my studies.

To my wife Renata: It is impossible to put into words all the gratitude I feel for your unwavering support and companionship during this period. I love you! To my parents, Marta and Gilson, I deeply appreciate the support throughout the many years I was away. I know it was not easy being half a world apart during most of this period. I love you! I would also like to thank my extended family, Nilton, Rosana, and Thiago, and all the other family members and friends that we missed dearly throughout these long years abroad.

To the many friends we made during this time: We will miss you dearly and hope to see you again soon. I would also like to thank all past and current members of the UW-Madison

Jiu-Jitsu Club for keeping me grounded and giving me another opportunity to develop my teaching skills.

The material in this dissertation is based upon work supported by the University of Wisconsin – Madison Office of the Vice Chancellor for Research and Graduate Education with funding from the Wisconsin Alumni Research Foundation, and by the U.S. National Science Foundation under Award No. 2312540. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

List of Notation

In the following, $\mathcal{C}$ denotes a generic complexity class.

AM	The class of languages accepted by polynomial-time Arthur-Merlin protocols (p. 44).
$\mathcal{A}\mathsf{SPCOMP}[s(n), \gamma(n)]$	The class of space-$s(n)$ bottleneck algorithms with compression length $\gamma(n)$ where the second phase is an algorithm of type $\mathcal{A}$ (p. 195).
$\mathcal{A}\mathsf{TICOMP}[t(n), \gamma(n)]$	The class of time-$t(n)$ bottleneck algorithms with compression length $\gamma(n)$ where the second phase is an algorithm of type $\mathcal{A}$ (p. 153).
$\mathsf{AMTIME}[t(n)]$	The class of languages accepted by time-$t(n)$ Arthur-Merlin protocols.
BPNL	A logspace version of AM (p. 208).
$\mathsf{BP} \cdot (\mathsf{UL} \cap \mathsf{coUL})$	The class of languages decidable by bounded-error logspace unambiguous algorithms that have an accepting computation path for every input.
BPP	The class of languages decidable by bounded-error probabilistic polynomial-time algorithms (p. 7).
$\mathsf{BPTIME}[t(n)]$	The class of languages decidable by bounded-error time-$t(n)$ probabilistic algorithms.

$\mathcal{C}/a(n)$	The class of languages decidable by algorithms of type $\mathcal{C}$ that receive $a(n)$ bits of additional non-uniform advice.
$\mathcal{C}^{\mathsf{prAM}}$	The class of languages decidable by algorithms of type $\mathcal{C}$ with oracle access to a prAM-complete problem.
$\mathcal{C}_{\|\|}^{\mathsf{prAM}}$	The class of languages decidable by algorithms of type $\mathcal{C}$ that can make non-adaptive oracle queries to a prAM-complete problem.
$\mathcal{C}^{\mathsf{SAT}}$	The class of languages decidable by algorithms of type $\mathcal{C}$ with oracle access to SAT.
$\mathcal{C}_{\|\|}^{\mathsf{SAT}}$	The class of languages decidable by algorithms of type $\mathcal{C}$ that can make non-adaptive oracle queries to SAT.
CL	The class of languages decidable by logspace catalytic algorithms (p. 195).
CNL	The class of languages decidable by nondeterministic logspace catalytic algorithms (p. 195).
co$\mathcal{C}$	The class of languages whose complements are in $\mathcal{C}$.
CUL	The class of languages decidable by unambiguous logspace catalytic algorithms (p. 195).
CUSPCOMP$[s(n), \gamma(n)]$	The class of space-$s(n)$ bottleneck algorithms with compression length $\gamma(n)$ where the second phase is an unambiguous logspace catalytic algorithm (p. 195).
DTIME$[t(n)]$	The class of languages decidable by time-$t(n)$ deterministic algorithms.
E	The class of languages decidable by linear-exponential-time deterministic algorithms.
EXP	The class of languages decidable by exponential-time deterministic algorithms.

FUL	The class of functions computable by unambiguous logspace algorithms (p. 194).
FCUL	The class of functions computable by unambiguous catalytic logspace algorithms (p. 194).
Heur	Heuristic/average-case modifier for a complexity class (p. 51).
io-$\mathcal{C}$	For $\mathcal{C}'$ a complexity class, normally used as in $\mathcal{C}' \subseteq$ io-$\mathcal{C}$. Indicates that for all $L' \in \mathcal{C}'$, there exists $L \in \mathcal{C}$ that agrees with L' on infinitely many input lengths.
L	The class of languages decidable by logspace algorithms.
MA	The class of languages accepted by polynomial-time Merlin-Arthur protocols.
MATIME$[t(n)]$	The class of languages accepted by time-$t(n)$ Merlin-Arthur protocols.
NC^1	The class of languages decidable by bounded fan-in logarithmic-depth circuits.
NE	The class of languages decidable by linear-exponential-time nondeterministic algorithms.
NEXP	The class of languages decidable by exponential-time nondeterministic algorithms.
NL	The class of languages decidable by nondeterministic logspace algorithms.
NP	The class of languages decidable by polynomial-time nondeterministic algorithms (p. 2).
$\mathsf{NP}^{\mathsf{NP}}$	The class of languages decidable by polynomial-time nondeterministic algorithms with oracle access to an NP-complete problem.

NP/poly	The class of languages decidable by polynomial-size nondeterministic circuits.
NTIME$[t(n)]$	The class of languages decidable by time-$t(n)$ nondeterministic algorithms.
P	The class of languages decidable by polynomial-time deterministic algorithms.
Π_2E	Equals coΣ_2E.
Π_2P	Equals coΣ_2P.
P^{NP}	The class of languages decidable by polynomial-time deterministic algorithms with oracle access to an NP-complete problem.
P/poly	The class of languages decidable by polynomial-size deterministic circuits.
prAMTICOMP$[t(n), \gamma(n)]$	The class of time-$t(n)$ bottleneck algorithms with compression length $\gamma(n)$ where the second phase is a prAM protocol (p. 153).
prBPTICOMP$[t(n), \gamma(n)]$	The class of time-$t(n)$ bottleneck algorithms with compression length $\gamma(n)$ where the second phase is a probabilistic algorithm (p. 153).
pr$\mathcal{C}$	The class of promise problems in the class $\mathcal{C}$.
$\mathsf{P}^{\Sigma_2\mathsf{EXP}}$	The class of languages decidable by polynomial-time deterministic algorithms with oracle access to a Σ_2EXP-complete problem.
PSPACE	The class of languages decidable by polynomial-space deterministic algorithms.
RP	The class of languages decidable by probabilistic polynomial-time algorithms with one-sided error.

SAT	The NP-complete Boolean satisfiability problem: Given a Boolean formula ϕ, determine if there is an assignment that satisfies ϕ.
Search–prBP $\cdot$ (UL $\cap$ coUL)	The search-problem version of the class prBP $\cdot$ (UL $\cap$ coUL) (p. 200).
Search–pr(UL $\cap$ coUL)	The search-problem version of the class pr(UL $\cap$ coUL) (p. 200).
Σ_2E	The class of languages decidable by linear-exponential-time nondeterministic algorithms with oracle access to an NP-complete problem.
Σ_2EXP	The class of languages decidable by exponential-time nondeterministic algorithms with oracle access to an NP-complete problem.
Σ_2P	Equals $\mathsf{NP}^{\mathsf{NP}}$.
Σ_2TIME$[t(n)]$	The class of languages decidable by time-$t(n)$ nondeterministic algorithms with oracle access to an NP-complete problem.
SIZE$[s(n)]$	The class of languages decidable by size-$s(n)$ deterministic circuits.
SPACE$[s(n)]$	The class of languages decidable by space-$s(n)$ deterministic algorithms.
TC^0	The class of languages decidable by unbounded fan-in constant-depth circuits with threshold gates.
UL	The class of languages decidable by unambiguous logspace algorithms.
UL/poly	The class of languages decidable by unambiguous logspace algorithms with polynomial-length advice.

$\mathsf{USPCOMP}[s(n), \gamma(n)]$	The class of space-$s(n)$ bottleneck algorithms with compression length $\gamma(n)$ where the second phase is an unambiguous logspace algorithm (p. 195).
ZPP	The class of languages decidable by zero-error probabilistic polynomial-time algorithms.
$\mathsf{ZPP}^{\mathsf{NP}}$	The class of languages decidable by zero-error probabilistic polynomial-time algorithms with oracle access to an NP-complete problem.

Chapter 1

Introduction

What is a mathematical proof? A common interpretation is as a sequence of symbols that when read by another person, probably a mathematician, convinces them that a certain statement is true. A more formal interpretation casts the proof verification process as a computation. Given a sequence of symbols representing the proof, the computation involves verifying that the sequence follows a previously defined set of rules: the axioms and rules of inference in a formal system.

When defining proofs, mathematicians may not take the length of the proof or the time required to verify it into consideration, but computer scientists do. After all, how useful is a proof that requires a thousand years to be verified? With the intent of defining "reasonable" proofs, we lay out some definitions. We say that a *decision problem* or Π is a subset of strings in some fixed alphabet, which for simplicity we set to $\{0, 1\}$. An *instance* of a decision problem is a binary string, which may be *positive* if it is in Π or *negative* otherwise. The definition of a decision problem allows for capturing true mathematical statements without proof length or verification time considerations: We just take Π to be the set of true mathematical statements with respect to some fixed formal system F.

The theory of computational complexity theory allows for a more grounded definition via the complexity class NP, where we limit the length of the proof as well as the time required

by the verification process. In the field of computational complexity, polynomial running times, those that grow polynomially with respect to the input length, are often considered "reasonable", and so we stick to this convention. The class NP consists of those decision problems Π for which there exists a polynomial-time algorithm V, called the verifier, such that:

- If x is a positive instance, then there exists a polynomial-length proof y such that $V(x, y)$ accepts.
- If x is a negative instance, then for all polynomial-length candidate proofs y', $V(x, y')$ rejects.

To formally capture "short" proofs in NP, we can define the following decision problem Π: Given a mathematical assertion ϕ in some fixed formal system F and an integer n in unary, is there a proof of length at most n in F for ϕ?

A more general interpretation of the notion of proof is that it is any process that persuades someone that a certain statement is true. Such a process can be a conversation: A *prover* tries to convince a *verifier* of the validity of the statement. Along the way, the verifier can ask questions until reaching a conclusion. The verifier may accept, indicating that they believe in the validity of the statement, or reject, indicating that they were not convinced by the prover.

At first, the class NP seems to disregard the conversational aspect of the more general interpretation of a mathematical proof. However, it turns out that NP does capture such interactions: An equivalent definition of NP is that of a computational process in which a computationally unbounded prover P interacts with a polynomial-time verifier V for as many rounds as necessary (though limited by the running time for the verifier) until the verifier accepts or rejects, with the requirement that some algorithm P representing a proof strategy that makes the verifier accept must exist for positive instances, and that no such

strategy can exist for negative instances. The reason for this equivalence is that a transcript for the interaction between P and V serves as a proof in the more traditional sense.

There is, however, one aspect that the class NP does not consider: randomness. What if we allow the verifier access to a fair coin to aid in the verification process? Do we gain anything from this change? To explore this notion, we discuss how randomness allows a person who can see colors — call him Merlin — to prove to a colorblind person — call him Arthur — that the two socks Arthur holds in his hands are of different colors — say red and green. The interaction between Merlin and Arthur goes as follows: First, Arthur secretly flips a coin. If the coin lands heads, Arthur shows Merlin the sock that he claims to be red, otherwise, Arthur shows Merlin the supposedly green sock. Merlin then tells Arthur whether the sock he sees is red or green. If the socks are indeed of different colors, Merlin will always be able to answer Arthur's challenge correctly. Otherwise, Merlin can't do better than just randomly guessing, answering incorrectly with probability exactly 1/2. Of course, there is still a probability of 1/2 on an individual interaction as above that Merlin can convince Arthur that the two socks are of different colors, even if they are not. However, repeating this procedure say, 10 times, already reduces this probability to 1/1024. In that case, even though the proof may not be satisfactory from a mathematical perspective, we may say that if Merlin always gets it right, Arthur is convinced with high probability that the socks are indeed of different colors.

The notion of a probabilistic proof as above can also be formalized by the theory of computational complexity, and we do so next.

Probabilistic proofs. Let Π be a decision problem. A *probabilistic proof* for Π is composed of two probabilistic algorithms V (for verifier) and P (for prover), together with a protocol that defines how the two entities interact. The interactive protocol must respect the following properties:

1. *Completeness.* For a positive instance of Π, the probability that V accepts after

interacting with P is 1, i.e., if the statement is true, then P can always make V accept the validity of the statement.

2. *Soundness.* For a negative instance of Π, for all algorithms P^*, the probability that V accepts after interacting with P is at most $1/2$, i.e., if the statement is false, then no proof strategy P^* can make V accept with high probability.

To guarantee that the definition above is as "reasonable" or efficient as the definition of NP, we also require that V runs in probabilistic polynomial time and that V and P exchange at most a polynomial number of messages, each of polynomial length. Notice that we do not limit the computational power for the prover P.

One may now ask: Are there any problems that admit efficient probabilistic proofs as above, and are not known to be in NP? The answer is yes, since efficient probabilistic proofs are captured by PSPACE, the class of problems decidable in polynomial *space* instead of time [Sha92]. As PSPACE is conjectured to be much larger than NP, introducing randomness seems to greatly increase the set of statements that can be proven. The class of problems admitting efficient probabilistic proofs is very robust. For example, we may allow the verifier to incorrectly reject positive instances with low probability, and also let the verifier reveal all of their coin tosses to the prover, without changing the class [FGM+89; GS86].

Having defined efficient probabilistic proofs and seen how powerful they are, we may still wonder if there is some restriction/subclass that is closer to NP while still allowing for "short" proofs for problems that are not known to be in NP. Looking ahead, we also want to find some subclass that could reasonably equal NP under some sort of *derandomization* hypothesis. In computational complexity, the subfield of derandomization asks whether it is possible to efficiently reduce or completely remove randomness from computational processes. In a sense, we are trying to increase the power of NP by introducing randomness, but then looking for ways to derandomize the resulting class. It turns out that restricting the number of rounds of communication to some constant independent of the input length

is enough. Before we define the resulting subclass, let us see an example of such a protocol, which has a very similar structure to the "colored-sock" protocol above, for a problem not known to be in NP.

The Graph Isomorphism problem is defined as follows: The input is a pair of graphs (G_0, G_1), and the objective is to decide whether G_0 is *isomorphic* to G_1. We say that two graphs are isomorphic if they have the same connectivity structure, even though they may look different at first. Formally, this means that there exists a permutation of one graph's vertices that maps it to the other graph (and vice-versa). Notice that Graph Isomorphism is in NP since a permutation as above serves as a short proof that can be efficiently verified. However, we don't know of any short proofs for negative instances: those pairs (G_0, G_1) of *nonisomorphic* graphs. Formally, we don't know whether the decision problem Graph Nonisomorphism is in NP, where Graph Nonisomorphism is the set of pairs of graphs that are *not* isomorphic. We do know, however, of an efficient probabilistic proof for Graph Nonisomorphism, which works as follows: For a pair of graphs G_0, G_1,

1. The verifier V samples a random bit b and a random permutation τ. V then applies τ to G_b, obtaining a graph H, which is sent to the prover P.

2. If H is isomorphic to G_0, P sets $b' = 0$, otherwise, it sets $b' = 1$. P then sends b' to V.

3. V accepts if and only if $b = b'$, i.e., P was able to detect which graph was randomly permuted to obtain H.

The key property in this protocol is that when G_0 is not isomorphic to G_1, a computationally-unbounded prover P can always discover to which of the two graphs H is isomorphic (for example, by trying out all possible permutations and checking which one maps H into one of G_0 or G_1), and can therefore make V accept with probability 1. However, when G_0 is isomorphic to G_1 (a negative instance for the problem Graph Nonisomorphism), the distribution of H is identical in the cases $b = 0$ and $b = 1$, meaning that no proof strategy

P^* can extract from H (the only message sent by V) the value of b with probability higher than 1/2. Thus, V rejects with probability exactly 1/2.

In the protocol above, it is critical that the verifier keeps the random bits it selects private, not revealing them to the prover. Otherwise, the prover would always be able to correctly guess which graph was randomly permuted by the verifier. However, there exists a protocol for Graph Nonisomorphism where it is safe for the verifier to reveal its random bits. The protocol has the following structure:

1. First, the verifier selects a polynomial number of random bits, obtaining a random string r, and sends r to the prover.

2. The prover replies with a proof y_r that may depend on the sequence r.

3. The verifier runs a deterministic verification process on the original input, r and y_r, accepting or rejecting.

Notice that in the protocol above, all of the verifier's random bits are revealed to the prover. A protocol with the structure above exists for all problems that admit constant-round probabilistic proofs, even ones where the verifier's randomness is private [GS86; BM88]. We call such a protocol an Arthur-Merlin protocol, a term coined by Babai and Moran [BM88].

Arthur-Merlin protocols. The idea behind the name Arthur-Merlin is that Merlin, an all-powerful but untrustworthy entity, wants to convince Arthur, a reliable but computationally-limited entity, that a certain assertion is true. To do so, Arthur first flips a coin a couple of times in plain view of Merlin, who then replies with a proof that is verified by Arthur. We define the class AM as the decision problems that admit an Arthur-Merlin protocol. By the discussion above, the problem Graph Nonisomorphism is in AM. We remark that there also exists the related class MA, where Merlin goes first in the protocol. It is known that NP is in MA, which is in AM, though it is unknown whether AM equals MA, or even whether Graph Nonisomorphism is in MA.

The class AM is the main topic of this dissertation. Our focus is on the following question that we alluded to earlier: Is AM equal to NP? Alternatively, can we remove all randomness in an Arthur-Merlin protocol at only a polynomial cost in time? As mentioned before, this is a derandomization question, and it turns out that there is ample evidence that AM equals NP. Next, we explore the derandomization subfield of computational complexity.

Derandomizing probabilistic algorithms. The classic derandomization question is concerned with efficient computation instead of efficiently-verifiable proofs, i.e., with the class BPP, which we define next. We say that a decision problem Π is in BPP if there exists a probabilistic polynomial-time algorithm A such that, for every instance x of Π, the following holds:

- If x is a positive instance, then the probability over r that $A(x;r)$ accepts is at least $2/3$.
- If x is a negative instance, then the probability over r that $A(x;r)$ rejects is at least $2/3$.

Above, we view the internal randomness of A as an additional input.

The main question posed by the field of derandomization is whether the class BPP equals P, the class of problems decidable by polynomial-time algorithms. Similar to what we briefly mentioned for AM, there is plenty of evidence suggesting that BPP equals P.

Because the algorithm A in the definition of BPP above runs in polynomial time, it can only access polynomially-many random bits during its execution. On an input x of length n, we can deterministically enumerate the $2^{\mathsf{poly}(n)}$ possible random strings r and compute exactly the probability that $A(x;\cdot)$ accepts, which implies that BPP is in EXP, the class of problems decidable in exponential time. The containment $\mathsf{BPP} \subseteq \mathsf{EXP}$ represents what we refer to as a trivial derandomization for BPP. A relaxed goal for the full derandomization $\mathsf{BPP} = \mathsf{P}$ is to obtain a faster-than-trivial deterministic algorithm for BPP: Even a subexponential-time

simulation remains open, where subexponential time refers to $\cap_{\epsilon>0}\mathsf{DTIME}[2^{n^\epsilon}]$. The latter setting is commonly referred to as the low end of the derandomization spectrum, while full derandomization is referred to as the high end.

How could we hope to figure out, deterministically, whether x is positive or negative without cycling through the values of r? A common approach is to employ a pseudorandom generator (PRG): A deterministic algorithm that produces a set of strings that "looks random" to any efficient computational process. To be more precise, a pseudorandom generator, on input a length n, outputs a multi-set S of strings such that, for any circuit C of size n, the following holds:

$$\left| \Pr_{r\in\{0,1\}^n}[C(r) = 1] - \Pr_{s\in S}[C(s) = 1] \right| \leq \epsilon,$$

where ϵ is a small constant. For derandomizing BPP, it suffices to set $\epsilon < 1/6$ to distinguish between positive and negative instances. This is the case because the algorithm A in the definition of BPP accepts or rejects correctly with probability $2/3$, and thus introducing an error $\epsilon < 1/6$ would not push the acceptance/rejection probability to $1/2$ or less.

We can capture the computation $A(x;\cdot)$ as a linear-size circuit, by padding if necessary. Thus, the existence of a pseudorandom generator computable in deterministic time $\mathsf{poly}(n)$ implies that BPP equals P: To derandomize an algorithm A on input x, the deterministic simulation D computes the multi-set S output by the pseudorandom generator. Then, for every $r \in S$, D computes $A(x;r)$. In the end, D accepts if $A(x;r)$ accepts for most r, and rejects if $A(x;r)$ rejects for most r. By a probabilistic argument, a random multi-set $S \subseteq \{0,1\}^n$ of size n is a pseudorandom generator with very high probability. The difficult part is computing such a multi-set deterministically.

So now we know of a potential way to derandomize BPP, but do pseudorandom generators exist? A long line of works connects the existence of efficiently-computable pseudorandom generators to circuit lower bounds for linear-exponential time. In particular, the assumption that the class $\mathsf{E} = \mathsf{DTIME}[2^{O(n)}]$ requires circuits of size at least $2^{\epsilon\cdot n}$ for some $\epsilon > 0$ implies the existence of a pseudorandom generator that suffices to show that BPP equals P [IW97].

Standard hierarchy theorems imply, for example, that $\mathsf{E} \notin \mathsf{DTIME}[2^{0.99 \cdot n}]$, and the only difference between $\mathsf{DTIME}[2^{0.99 \cdot n}]$ and $\mathsf{SIZE}[2^{0.99 \cdot n}]$ (the class of problems computable by circuits of size $2^{0.99 \cdot n}$) is that the latter class allows for a different algorithm for each input length. Since even a lower bound against the class $\mathsf{SIZE}[2^{0.01 \cdot n}]$ would suffice for derandomization, the circuit lower bounds that imply derandomization are considered believable, and so is derandomization. Weaker lower bounds, such as superpolynomial-size circuit lower bounds for E, imply weaker derandomization results (for the example, a low-end derandomization), and there is a smooth interpolation between the two extremes [Uma03].

We remark that the circuit lower bounds that are required for derandomization need to hold *almost-everywhere*: For every sufficiently large input length and every "small" circuit C, there must exist some input of that length where C fails to compute the corresponding decision problem Π. A less restrictive lower bound is one that holds *infinitely-often*, which only guarantees that there exist infinitely-many input lengths where every "small" circuit fails to compute Π. We address this technical distinction in the body but ignore it in this introduction.

At the same time that the circuit lower bounds mentioned above are believable, proving such results is complicated, and several barrier results indicate that doing so is beyond current techniques, such as the natural proofs barrier of Razborov and Rudich [RR97]. The discussion leaves a question open: Are pseudorandom generators, and thus circuit lower bounds, necessary for derandomization? Notice that pseudorandom generators imply a strong, *blackbox* derandomization: The set of strings output by a PRG "fools" every circuit of a given size. For certain parameter settings, in particular the low-end, it is known that derandomization implies the existence of *nondeterministic* pseudorandom generators with the parameters necessary for recovering the derandomization [IKW02]. Because such pseudorandom generators are nondeterministic instead of deterministic, these actually recover a nondeterministic simulation for BPP instead of a deterministic one.

Morally, pseudorandom generators seem like overkill for derandomization. For example,

instead of "fooling" circuits, which are a non-uniform model of computation, it would suffice to "fool" the algorithms $A(x;\cdot)$. Under a uniform hardness assumption such as $\mathsf{EXP} \not\subseteq \mathsf{BPP}$ [IW01], it is possible to construct a pseudorandom generator that "fools" algorithms $A(x;\cdot)$ on *average*, meaning that the pseudorandom generator "fools" $A(x;\cdot)$ over a random choice of *input* x of a certain length with high probability. At the low end of the derandomization spectrum, an equivalence is known between the assumption $\mathsf{EXP} \not\subseteq \mathsf{BPP}$ and a subexponential-time simulation for BPP. Such uniform hardness versus randomness results are still open for the high end of the derandomization spectrum (see [TV07; CRT+20; CRT22] for progress toward an equivalence at the high end).

Given the discussion above, we may ask whether there exists some type of relaxation for a pseudorandom generator that is equivalent to any type of derandomization that works on all inputs. Such derandomizations are called *whitebox* derandomizations in contrast to the *blackbox* derandomizations obtained via pseudorandom generators. For BPP, the answer is positive: Whitebox derandomization is equivalent to the existence of *targeted* pseudorandom generators [Gol11; Gol20]. These objects receive as input a circuit (representing a computation $A(x;\cdot)$, for example) and are only required to produce a set of strings that "look random" to that particular circuit. Until recently, a hardness assumption that is equivalent to the existence of targeted generators (and thus derandomization of BPP) was unknown. We discuss recent results that address this question in Section 1.1.

Before moving on to the AM setting, we remark that pseudorandom generators sufficient for derandomizing BPP also suffice for derandomizing the related class MA. The reason is that, after fixing an input x and Merlin's message, the remaining computation can be captured by a deterministic circuit that receives Arthur's randomness as input.

Derandomizing Arthur-Merlin protocols. In the AM setting, there are some similar results to those of the BPP setting, but in some cases changes are necessary. For example, it is known that AM is in EXP, since in exponential time we can not only enumerate Arthur's

randomness, but also all possible Merlin responses. We can also consider derandomizing Arthur-Merlin protocols using pseudorandom generators. Instead of selecting random bits, Arthur can compute the multi-set S output by the PRG, and execute, for each $s \in S$, a parallel instance of the interaction with Merlin by sending s to Merlin and receiving a proof y_s back. In the end, Arthur accepts if all proofs sent by Merlin were valid, i.e., would be accepted by the original protocol, and rejects if any of the proofs were invalid. The above simulation can be carried out in NP: The proof is composed of Merlin's messages. It is unknown, however, whether regular pseudorandom generators, such as the ones sufficient for derandomizing BPP, suffice for derandomizing AM. The reason is that if we view Arthur's randomness as an input for an AM protocol P, and fix an input x, the computation $P(x;\cdot)$ is captured by a *nondeterministic* circuit that guesses Merlin's response and simulates the final computation performed by Arthur. Since all we know about the PRG is that it "fools" *deterministic circuits*, we cannot guarantee that the simulation above works.

It is natural to consider a strengthening of the notion of pseudorandom generator so that it "fools" nondeterministic circuits. By the discussion above, such generators are sufficient for derandomizing AM. Because the objective is to obtain a nondeterministic simulation, it also suffices for the pseudorandom generator for nondeterministic circuits to be computable in nondeterministic polynomial time. Assuming that Merlin's objective is to make Arthur accept, the only mistake Arthur can make when interacting with Merlin is to accept a false statement, which happens with low probability. This is the reason the nondeterministic simulation above is sufficient for derandomization: If Merlin cannot produce a proof y_r for some message r sent by Arthur, then the statement must be false. In that case, a relaxation of a pseudorandom generator, called a *hitting-set generator*, suffices for derandomizing AM.

So now we know that pseudorandom generators for nondeterministic circuits are enough to obtain a blackbox derandomization for AM, but do they exist? Similar hardness versus randomness results to the ones that hold for BPP also hold for Arthur-Merlin protocols. For example, by observing that the Nisan-Wigderson pseudorandom generator construction

relativizes, Klivans and Van Melkebeek showed that linear-exponential SAT-oracle circuit lower bounds for $\mathsf{NE} \cap \mathsf{coNE}$, a nondeterministic analogue of E, imply that AM equals NP [KvM02]. Later works managed to weaken the assumption to nondeterministic circuit lower bounds, and even showed that lower bounds for classes such as E and $\mathsf{NE} \cap \mathsf{coNE}$ against nondeterministic circuits are equivalent to lower bounds against non-adaptive SAT oracle circuits [MV05; SU05; SU06]. As in the BPP setting, the lower bound equivalences for blackbox derandomization of AM scale smoothly. Also similar to the BPP setting, proving the circuit lower bounds that are equivalent to blackbox derandomization seems out of reach, and the only known equivalence between blackbox derandomization and general, whitebox derandomization is at the low-end of the derandomization, and one level up in the polynomial hierarchy [AvM17].

Uniform hardness versus randomness results are also known for AM. Under assumptions such as $\mathsf{EXP} \not\subseteq \mathsf{AMTIME}[2^{\epsilon n}]$ for some $\epsilon > 0$, one can conclude some average-case polynomial-time derandomization for AM [GST03]. Unlike the BPP setting, low-end uniform hardness versus randomness results for AM remain open, though some constructions nearly achieve such results [SU09].

In this work, we focus on the whitebox setting for Arthur-Merlin protocols, that is, on any derandomization that may, for example, inspect the specific protocol being derandomized. Until our work, not much was known about this setting for AM. For example, when Goldreich showed that BPP equals P is equivalent to the existence of targeted pseudorandom generators, he asked about a similar result in the AM setting [Gol11]. The main difference between the two settings, and the reason that obtaining such an equivalence for AM is complicated, is that the techniques underlying the result for BPP rely on treating polynomial-time algorithms as oracles. Since P equals coP, the class of decision problems whose complements are in P, replacing the oracles by actual polynomial-time algorithms still results in a polynomial-time algorithm. For AM, the technique presents an issue since we don't know whether NP equals coNP, and thus we must work harder in the search for

equivalences.

In the following sections, we provide a summary of the main results of this dissertation, together with additional context and motivation. The main body of this dissertation further expands on the concepts and formally develops the assertions presented in this chapter. We now present our results on derandomization of Arthur-Merlin protocols.

1. We obtain near-equivalences between whitebox derandomization of Arthur-Merlin protocols and the existence of multi-bit functions that are computable in nondeterministic polynomial time and hard on almost-all inputs against Arthur-Merlin protocols. We also take a first step toward a resolution for Goldreich's question, and obtain other byproducts regarding derandomization of Arthur-Merlin protocols. These contributions appear in Chapter 2.

2. We obtain full hardness versus randomness equivalences for *mild* derandomization of Arthur-Merlin protocols, i.e., simulations in classes that are supposedly larger than NP. The equivalences are with respect to the existence of multi-bit functions that remain hard to compute even in the presence of efficiently-computable leakage of their value. Along the way, we resolve Goldreich's question in the affirmative in the mild setting: Mild derandomization of Arthur-Merlin protocols is equivalent to the existence of targeted pseudorandom generators. These contributions appear in Chapter 3.

3. We characterize the existence of targeted pseudorandom generators for Arthur-Merlin protocols in terms of a constructive hardness assumption: We show that derandomization via targeted generators is equivalent to the existence of refuters for functions computable in nondeterministic polynomial time against Arthur-Merlin protocols that go through a compression phase. Along the way, we also generalize our results in the mild setting. These contributions appear in Chapter 4.

Though our main focus is on derandomization of *time-bounded* Arthur-Merlin *protocols*, we also develop results for *space-bounded algorithms*. We obtain conditional results in the

setting of space-bounded isolation and equivalences for derandomization of unambiguous space-bounded algorithms. We also show that a space-bounded version of the class AM can be decided in deterministic catalytic logspace. These contributions appear in Chapter 5.

1.1 Instance-wise hardness

We present our first steps in the whitebox derandomization setting for AM in Chapter 2. For future reference and comparison, we start with a high-level description of the recent results in the whitebox derandomization setting for BPP.

As mentioned earlier, whitebox derandomizations for BPP that work on all inputs are equivalent to the existence of targeted pseudorandom generators. Chen and Tell [CT21] proposed an equivalent lower bound condition: uniform lower bounds for multi-bit functions f that hold on almost-all inputs, i.e., any algorithm in the class for which the lower bound holds can only compute f on finitely many inputs.

Chen and Tell first observe that, by diagonalization, the derandomization assumption $\mathsf{BPP} = \mathsf{P}$ implies that there exists a multi-bit function computable in deterministic polynomial time that is hard on almost-all inputs against "faster" probabilistic algorithms. More importantly, they establish an almost-converse: If there exists a multi-bit function computable by logspace-uniform shallow circuits that is hard on almost-all inputs against "faster" probabilistic algorithms, then $\mathsf{BPP} = \mathsf{P}$. The main gap between the two implications is the low-depth requirement for the direction of hardness to derandomization.

The Chen-Tell transformation of hardness to derandomization is *instance-wise*: This means that the derandomization result they obtain works for every input x for which a certain algorithm (looking ahead, their reconstructor) fails to compute $f(x)$. This is why hardness on almost-all input implies derandomization that works for every sufficiently large input ($\mathsf{BPP} = \mathsf{P}$). Similarly, one may assume that their reconstructor fails to compute $f(x)$ for most inputs of each length and obtain an average-case derandomization, for example.

The main technical contribution of the Chen-Tell result is a conditional targeted pseudorandom generator construction based on *learning* reconstructors for hardness-based pseudorandom generator constructions. To understand the concept, we first describe the usual proofs of correctness for the classical pseudorandom generator constructions. Recall that a hardness-based PRG construction relies on circuit lower bounds. To show correctness of such a construction, one shows that if the multi-set output by the PRG based on function f is not pseudorandom, i.e., there exists some small *distinguisher* circuit D that is not fooled by the PRG, then there exists a small circuit computing f. Specifically, most constructions show something stronger: There exists an efficient *learning* algorithm that when given oracle access to f and to the distinguisher D, outputs a small circuit computing f. We call such an algorithm a *reconstructor*.

Chen and Tell combine the Nisan-Wigderson (NW) generator [NW94] with the doubly-efficient proof system of Goldwasser, Kalai and Rothblum [GKR15]. Let C be a uniform low-depth circuit computing the hard function f and fix an input x. The GKR proof system captures the computation of C on input x into a sequence of polynomials such that computing a polynomial in the sequence is efficiently reducible to computing the previous polynomial. The polynomials represent layers in the the computation of C on input x, and thus the first polynomial is easily computable given x, and the last polynomial contains the value of $f(x)$. On input x, the Chen-Tell generator applies the NW generator to each polynomial in the sequence. In case the generator fails, the learning property of the NW reconstructor allows them to efficiently produce a small circuit computing the i-th polynomial, given oracle access to the i-th polynomial. The reducibility properties for the polynomials in the sequence allows them to start from a trivial circuit computing the first polynomial, and bootstrap the construction of circuits for each polynomial in the sequence, ending with the polynomial that captures $f(x)$. Assuming the depth for the circuit C is small, the process leads to a faster algorithm for computing f.

Follow-up works managed to obtain full equivalences for derandomization of BPP, namely

with respect to hardness of a computational problem related to Levin-Kolmogorov complexity [LP22] and hardness in the presence of efficiently-computable leakage [LP23]. We explore the latter in more detail in Chapter 3.

Contributions We start by observing that hardness on almost-all inputs is required for derandomizing AM. To be a bit more precise, we show that if $\mathsf{AM} = \mathsf{NP}$, then there exists a multi-bit function f that is computable in nondeterministic polynomial time with "a few" bits of advice, and is hard on almost-all inputs against "faster" Arthur-Merlin protocols.

As our main result for the chapter, we obtain an almost-converse, that hardness on almost-all inputs is sufficient for derandomizing AM. Specifically, we show that if there exists a multi-bit function f computable in nondeterministic polynomial time that is hard on almost-all inputs against "faster" promise Arthur-Merlin protocols, then $\mathsf{AM} = \mathsf{NP}$. Just like the Chen-Tell result, our result is instance-wise: Derandomization holds for each input where the hardness assumption with relation to our reconstructor holds. In the setting of hardness on almost-all inputs, the remaining gaps between the two directions are the advice and the technical difference between regular and promise Arthur-Merlin protocols.

For the main result of the chapter, we combine probabilistically-checkable-proofs (PCPs) with a recursive variant of the Miltersen-Vinodchandran generator due to Shaltiel and Umans [SU09], which we dub RMV. Combining these ingredients, we obtain a conditional targeted pseudorandom generator construction. To derandomize an Arthur-Merlin protocol P on an input x, given a hard function f, the generator guesses and verifies a PCP asserting the value of $f(x)$, and uses the PCP as a basis for the RMV generator. In case the generator fails to "fool" $P(x;\cdot)$, the RMV reconstructor allows for compressing the PCP using an Arthur-Merlin protocol, which leads to a speed-up for computing f, as the PCP verifier is very efficient.

As byproducts of our targeted pseudorandom generator construction, we obtain new results in the uniform setting, concluding derandomization that works on average (with high

probability over a random input) from worst-case hardness.

Acknowledgment of contributions. The results in Chapter 2 appear for the most part in CCC 2023 [vMM23a], and represent joint work with Dieter van Melkebeek.

1.2 Mild derandomization

As mentioned previously, there is evidence that $\mathsf{AM} = \mathsf{NP}$. However, we don't even know, for example, whether $\mathsf{AM} \subseteq \Sigma_2\mathsf{P} = \mathsf{NP}^{\mathsf{NP}}$, i.e., we don't know whether an extra layer of non-determinism is enough to eliminate randomness of Arthur-Merlin protocols at a polynomial slowdown. We refer to this setting as the *mild* derandomization setting, and mention that the inclusion above is open even for subexponential-time simulations.

Chapter 3 continues the study of whitebox derandomization for AM in the search of equivalences, though this time in the *mild* setting. Our hope is that the relaxed derandomization requirement allows us to obtain equivalences which can be later leveraged into the standard setting.

Contributions. As our main contribution for the chapter, we fully characterize mild derandomization for AM protocols in terms of leakage-resilient hardness. As in [LP23], we say that a function f is leakage-resilient hard on almost all inputs against a class $\mathcal{C}$ if any pair of algorithms (Leak, A) in $\mathcal{C}$ that operate as follows can only compute f with high probability on finitely many inputs.

1. The algorithm Leak on input $x \in \{0,1\}^n$ and the value of $f(x)$, outputs a short string π, say of length $\sqrt{n}$.

2. The algorithm A, on input x and π, attempts to compute $f(x)$.

Our result reads as follows: $\mathsf{AM} \subseteq \Sigma_2\mathsf{P}$ if and only if there exists a multi-bit function f computable in $\Sigma_2\mathsf{P}$ that is leakage-resilient hard on almost-all inputs against non-adaptive

SAT-oracle algorithms. As mentioned earlier, hardness against non-adaptive SAT-oracle circuits is equivalent to blackbox derandomization for AM. We show a similar equivalence for the whitebox mild derandomization setting. Our equivalence scales throughout the entire derandomization spectrum by varying the time required for computing the hard function f, and also applies to intermediate classes between NP and Σ_2P such as P^{NP} and $\mathsf{ZPP}^{\mathsf{NP}}$.

To obtain our main result for the chapter, we develop a *learning* reconstructor for the nondeterministic pseudorandom generator construction due to Shaltiel and Umans, which we denote by SU [SU05]. Specifically, we show that the SU reconstructor can be cast as an efficient algorithm that, given oracle access to a function g, outputs with high probability a small non-adaptive SAT-oracle circuit that computes g. Given a leakage-resilient hard function f and an input x, we construct a targeted pseudorandom generator that uses the value of $f(x)$ as a basis for the SU generator, i.e., it uses the function g that maps i to the i-th bit of $f(x)$ as the basis for SU. If the derandomization on input x fails, then there exists an efficient algorithm Leak that, on input x (which is necessary to describe the distinguisher for the SU generator) and $f(x)$, outputs a small non-adaptive SAT-oracle circuit that computes the mapping $i \mapsto f(x)_i$. It then suffices to set A as the algorithm that, on input a circuit C, outputs the truth-table of C.

As a consequence of the characterization via leakage-resilient hardness, we also show that mild derandomization for AM is equivalent to the existence of targeted pseudorandom generators recovering the same simulation, thus resolving Goldreich's question [Gol11] in the affirmative for mild simulations.

Finally, we show that the hardness assumption can be relaxed to that of hardness on almost-all inputs against *learn-and-evaluate protocols* (See Section 2.3.3 for a definition), and connect leakage-resilience to nondeterministic circuit lower bounds at the low end of the mild derandomization spectrum.

In Chapter 2, as an additional by-product of our conditional targeted pseudorandom generator construction, we also obtain an unconditional average-case subexponential-time

mild derandomization for AM with subpolynomial advice.

Acknowledgment of contributions. The results in Chapter 3 appear for the most part in FSTTCS 2023 [vMM23b], and represent joint work with Dieter van Melkebeek.

1.3 Refuting bottleneck protocols

Recall that, in the leakage-resilient hardness setting and to compute a function f, we have a pair of algorithms (Leak, A) that work as follows: Leak gets $f(x)$ as additional input and outputs a small amount of leakage π on the value of $f(x)$. Then, the algorithm A, on input x and π, attempts to compute $f(x)$.

Consider the process of, after fixing a pair (Leak, A) as above and an input x, finding a value of $f(x)$ such that (Leak, A) fails to compute $f(x)$ with high probability. Because of the *bottleneck* in this process, an information-theoretic argument guarantees that, for sufficiently large n, the pair (Leak, A) fails to recover $f(x)$ for the great majority of inputs $f(x)$ of length n. We may take a step further and view the process of computing $f(x)$ as a *refutation* task: Consider $\text{Leak}(x, \cdot)$ as a compression algorithm A_{comp}, and $A(x, \cdot)$ as a decompression algorithm A_{dec}. Our objective is then to find an input z such that $(A_{\text{comp}}, A_{\text{dec}})$ fail to recover z with high probability, i.e., to solve a refutation task for the identity function against such *bottleneck algorithms*.

A recent contribution of Chen, Tell and Williams [CTW23], shows that, if there is a deterministic polynomial-time algorithm that, on input 1^n and a pair $(A_{\text{comp}}, A_{\text{dec}})$ as above, outputs a string z such that $(A_{\text{comp}}, A_{\text{dec}})$ fails to recover z with high probability, then BPP = P. Their result is actually stronger in two ways:

- Their result establishes an equivalence with derandomization: If BPP = P, then there exists an algorithm as above.

- Viewing the algorithm above as a refuter for the identity function against bottleneck algorithms, the identity function can be replaced by any other function computable in deterministic polynomial time.

The Chen-Tell-Williams result captures a line of works that introduce a perspective shift in relation to constructing targeted pseudorandom generators [Kor22a; Kor22b; CJS+24]. Instead of relying on a hardness assumption that holds on almost-all inputs to obtain derandomization, it suffices to rely on a *constructive* hardness assumption for a function that is trivially hard for a very restrictive class of algorithms. Most hardness-based pseudorandom generator reconstructors can be cast as the compressing/decompressing process above, and thus *algorithmically* finding an input where this process fails leads to derandomization.

Our starting point for Chapter 4 is the question: How can we employ this new perspective in the AM setting?

Contributions. As our main contribution for the chapter, we fully characterize derandomization of AM through targeted pseudorandom generators in terms of refuters against *bottleneck protocols*. Such protocols operate in a similar way as the bottleneck algorithms above, where we replace A_{dec} by an Arthur-Merlin protocol P_{dec}.

Informally, we show that there exists a targeted pseudorandom generator achieving the derandomization $\mathsf{AM} = \mathsf{NP}$ if and only if there exists a refuter for the identity function computable in nondeterministic polynomial time against polynomial-time bottleneck protocols. We also show that the identity function can be replaced by any function computable in nondeterministic polynomial time, and our equivalence scales throughout the entire derandomization spectrum by varying the running time for the refuter.

To obtain our main result for the chapter, we adopt the new refutation perspective and refine the conditional targeted pseudorandom generator construction of Chapter 2 in the following ways:

- We replace PCPs in our original construction with probabilistically-checkable proofs

of proximity (PCPPs) and have the compressing algorithm encode the input with a suitable error-correcting code (ECC).

- We additionally employ the RMV generator with the input x as a basis, which leads to a bottleneck reconstructor when combined with the PCPP above.

For the direction of derandomization to refutation, we exploit a resilience property of our conditional targeted pseudorandom generator construction, which allows us to use a targeted pseudorandom generator for AM to find an input where the reconstructor for the generator fails.

As additional results, we obtain a characterization of mild derandomization in terms of refutation that generalizes our leakage-resilient hardness characterization of Chapter 3. We also connect nondeterministic circuit lower bounds for the class $\Sigma_2\mathsf{EXP}$ (the exponential-time analogue of $\Sigma_2\mathsf{P}$) to the existence of refuters against non-adaptive SAT-oracle bottleneck algorithms that compress their input to a string of polylogarithmic length. Finally, we explore consequences of derandomizing Arthur-Merlin protocols to explicit constructions of combinatorial objects.

Acknowledgment of contributions. The results in Chapter 4 represent joint work under review with Dieter van Melkebeek.

1.4 Space-bounded computation

Whereas the previous results are in the setting of time-bounded Arthur-Merlin protocols, we now consider space-bounded computations. We focus on two settings for space-bounded computations: *isolation* and *catalytic computation*.

Isolation is the process of singling out solutions for computational problems that may have multiple solutions. For the class NL, of computational problems decidable by nondeterministic algorithms in logarithmic space, isolation is equivalent to the existence of an

unambiguous logspace algorithm deciding the NL-complete problem of reachability for directed graphs. An unambiguous algorithm is a nondeterministic algorithm that either has a single accepting computation path or none. The existence of such an algorithm would imply that NL is equal to the class UL of logspace unambiguous algorithms.

Due to the Isolation Lemma, the problem of space-bounded isolation is essentially a derandomization problem [RA00; vMP19]: Assigning small random weights to the edges in a directed graph results in a weighted graph with unique shortest paths between every pair of vertices.

Catalytic algorithms are algorithms that have access to a small regular read-write memory, while also having access to an additional, larger read-write memory. The additional memory is already populated with arbitrary data, which the algorithm is allowed to modify but must restore to its original content before the end of the execution.

Contributions. By leveraging the new perspective of Chapter 4 as well as recent space-efficient reconstructors for the NW generator [DT23; DPT24], we are able to deduce isolation in the space-bounded setting from refutation assumptions similar to those of Chapter 4, and in some cases obtain full equivalences. We also prove unconditionally that BPNL, a space-bounded version of the class AM, is contained in CL, the class of problems decidable by catalytic logspace algorithms.

Acknowledgment of contributions. The results in Chapter 5 represent unpublished, joint work with Dieter van Melkebeek.

Chapter 2

Instance-Wise Hardness

2.1 Introduction

We start with a more formal recap of the discussion of blackbox and whitebox derandomization for BPP and AM in Chapter 1.

BPP setting. The first hardness vs. randomness tradeoffs were developed for *blackbox* derandomization, where a pseudorandom generator (PRG) produces, in an input-oblivious way, a small set of strings that "look random" to the process under consideration on every input of a given length. A long line of research established tight equivalences between *blackbox* derandomization of prBPP (the promise version of the class BPP) and *nonuniform* lower bounds for exponential-time classes. At the low end of the derandomization spectrum, subexponential-time blackbox derandomizations of prBPP are equivalent to superpolynomial circuit lower bounds for $\mathsf{EXP} \doteq \mathsf{DTIME}[2^{\mathsf{poly}(n)}]$ [BFN+93]. At the high end, polynomial-time blackbox derandomizations of prBPP are equivalent to linear-exponential circuit lower bounds for $\mathsf{E} \doteq \mathsf{DTIME}[2^{O(n)}]$ [IW97]. A smooth interpolation between the two extremes exists and yields tight equivalences over the entire derandomization spectrum [Uma03]. The results are also robust in the sense that if the circuit lower bound holds at infinitely many input lengths (equivalent to the separation $\mathsf{EXP} \not\subseteq \mathsf{P/poly}$ at the low end),

then the derandomization works at infinitely many input lengths, and if the circuit lower bound holds at almost-all input lengths, then the derandomization works at almost-all input lengths.

A uniformization of the underlying arguments led to equivalences between derandomizations that work on *most* inputs of a given length, and *uniform* lower bounds, i.e., lower bounds against algorithms. This derandomization setting is often referred to as the *average-case* setting.[1] At the low end, there exist subexponential-time simulations of BPP that work on all but a negligible fraction of the inputs of infinitely many lengths if and only if $\mathsf{EXP} \not\subseteq \mathsf{BPP}$ [IW01]. Unfortunately, the known construction does not scale well (see [TV07; CRT+20; CRT22] for progress toward an equivalence at the high end) and is not robust (a version for almost-all input lengths remains open). On the other hand, the result holds for blackbox derandomization as well as for general, "whitebox" derandomization, and implies an equivalence between blackbox and whitebox derandomization in this setting: If derandomization is possible at all, it can be done through pseudorandom generators.

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

Definition 2.1 (Hardness on almost-all inputs). A computational problem f is hard on almost-all inputs against a class of algorithms if for every algorithm A in the class there is at most a finite number of inputs on which A computes f correctly. ◂

Chen and Tell started from the following observation about derandomization to hardness

[1] The underlying distribution may be the uniform one or any other polynomial-time sampleable distribution.

at the high end of the spectrum.

Proposition 2.2 (Chen and Tell [CT21]). *If* $\mathsf{prBPP} \subseteq \mathsf{P}$*, then for every constant* c *there exists a length-preserving function* f *that is computable in deterministic polynomial time and is hard on almost-all inputs against* $\mathsf{prBPTIME}[n^c]$*.*

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

Theorem 2.3 (Chen and Tell [CT21]). *Let* f *be a length-preserving function computable by logspace-uniform circuits of polynomial size and depth* n^b *for some constant* b*. If* f *is hard on almost-all inputs against* $\mathsf{prBPTIME}[n^{b+O(1)}]$*, where* $O(1)$ *denotes some universal constant, then* $\mathsf{prRP} \subseteq \mathsf{P}$*.*

Note that the hardness hypothesis of Theorem 2.3 necessitates the depth n^b of the uniform circuits computing the function f to be significantly less than their size. Otherwise, there exists even a deterministic algorithm that computes f in time $n^{b+O(1)}$.

The proof of Theorem 2.3 constructs a polynomial-time targeted hitting-set generator for prRP, which generically implies a polynomial-time targeted pseudorandom generator for prBPP, and thus that $\mathsf{prBPP} \subseteq \mathsf{P}$. Theorem 2.3 scales smoothly over the entire derandomization spectrum for prRP. Due to losses in the generic conversion from hitting sets to derandomizations for two-sided error, the corresponding result for prBPP does not scale that well. In particular, a low-end variant of Theorem 2.3 for prBPP remains open. That said, the results are robust in a similar sense as above with respect to input lengths. In fact, the approach inherently yields a much higher degree of robustness because it effectuates a hardness vs. randomness tradeoff on an input-by-input basis, as we explain further in the paragraph below about our techniques.

As a summary of the above discussion, Table 2.1 provides a qualitative overview of the lower bound equivalences for each of the three types of derandomization considered. We point out that, in the new setting of whitebox derandomizations that work on almost-all inputs, an actual equivalence along the lines of Chen and Tell [CT21] remains open due to the additional uniform-circuit depth requirement that is needed in the direction from hardness to derandomization. We refer to such results as *near-equivalences*. Follow-up works managed to obtain full-fledged equivalences in terms of other types of hardness, namely hardness of a computational problem related to Levin-Kolmogorov complexity [LP22] and hardness in the presence of efficiently-computable leakage [LP23].

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

Table 2.1: Equivalences between various types of derandomization and lower bounds

AM setting. An equivalence corresponding to the first line of Table 2.1 is known throughout the entire spectrum [KvM02; MV05; SU05]. The role of EXP is now taken over by $\mathsf{NEXP} \cap \mathsf{coNEXP}$, and the circuits are nondeterministic (or single-valued nondeterministic, or deterministic with oracle access to an NP-complete problem like SAT). The simulations use hitting-set generators for AM that are efficiently computable nondeterministically. Hitting-set generators are the natural constructs in the setting of AM because every Arthur-Merlin protocol can be efficiently transformed into an equivalent one with perfect completeness. As in the BPP setting, the lower bound equivalences for blackbox derandomization of prAM scale smoothly and are robust with respect to input lengths.

Regarding derandomizations that work on all but a negligible fraction of the inputs of a given length (the second line in Table 2.1), no hardness vs. randomness tradeoffs for AM were known prior to our work. What was known, are high-end results on derandomizations where no efficient nondeterministic algorithm can locate inputs on which the simulation is

guaranteed to be incorrect [GST03; SU09]. Indeed, the authors of [GST03] explicitly mention the average-case setting and why their approach fails to yield average-case simulations that are correct on a large fraction of the inputs. The setting corresponding to the third line in Table 2.1 was not studied before.

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

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

Proposition 2.4. *If* $\mathsf{prAM} \subseteq \mathsf{NP}$*, then for every constant* c *there exists a length-preserving function* f *that is computable in nondeterministic polynomial time with "a few" bits of advice, and is hard on almost-all inputs against* $\mathsf{AMTIME}[n^c]$*.*

We refer to Section 2.5.1 for the quantification of "a few".

Importantly, we are able to establish an almost-converse of Proposition 2.4. Under a slightly stronger hardness assumption, we construct a targeted hitting-set generator for prAM that is computable in nondeterministic polynomial time, yielding the following derandomization result.

Theorem 2.5. *Let* f *be a length-preserving function computable in nondeterministic time* n^a *for some constant* a*. If* f *is hard on almost-all inputs against* $\mathsf{prAMTIME}[n^c]$ *for* $c = O((\log a)^2)$*, where* $O(\cdot)$ *hides some universal constant, then*

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

Note that, in contrast to Theorem 2.3 in the BPP setting, Theorem 2.5 in the AM setting has no uniform-circuit depth restriction on the function f. Together with Proposition 2.4, Theorem 2.5 represents a near-equivalence between $\mathsf{prAM} \subseteq \mathsf{NP}$ and hardness on almost-all

inputs of length-preserving[2] functions against Arthur-Merlin protocols. Whereas in the BPP setting, the remaining gap relates to uniform-circuit depth, in the AM setting the remaining gap relates to the advice and the technical distinction between AM and prAM protocols. We remark that we do obtain full equivalences in the next chapters. In Chapter 3, we obtain full equivalences for mild derandomization, i.e., simulations on Σ_2-machines, and in Chapter 4 we obtain an equivalence with respect to the existence of targeted hitting-set generators for prAM.

Both Proposition 2.4 and Theorem 2.5 scale quite smoothly across the derandomization spectrum. The generalization of Theorem 2.5 has the following form: Let f be a length-preserving function computable in nondeterministic time $T(n)$. If f is hard on almost-all inputs against $\mathsf{prAMTIME}[t(n)]$, then $\mathsf{prAM} \subseteq \mathsf{NTIME}[\mathsf{poly}(T(n))]$. Intuitively, we may think of $t(n)$ as only slightly smaller than $T(n)$ for high-end results and much smaller for low-end results. Pushing our techniques as far as possible toward the low end, we obtain the following variant of Theorem 2.5.

Theorem 2.6. *Let f be a length-preserving function computable in nondeterministic exponential time. If f is hard on almost-all inputs against* $\mathsf{prAMTIME}[n^{b(\log n)^2}]$ *for all constants b, then for some constant c*

$$\mathsf{prAM} \subseteq \mathsf{NTIME}[2^{n^c}]. \tag{2.1}$$

As $\mathsf{prAM} \subseteq \mathsf{NEXP}$ trivially holds, the conclusion (2.1) of Theorem 2.6 represents the very low end of the derandomization spectrum. Note that a perfectly smooth scaling of Theorem 2.5 would only need a polynomial lower bound to arrive at the conclusion of Theorem 2.6, but the hypothesis of Theorem 2.6 requires a lower bound of $n^{\omega((\log n)^2)}$. We remark that the same discrepancy shows up in the current best-scaling uniform hardness vs.

[2]The focus on length-preserving functions f in Proposition 2.4 and Theorem 2.5 is for concreteness. For Proposition 2.4 to hold, the number of output bits needs to grow with n in an efficiently computable fashion. For Theorem 2.5 any number of output bits suffices as long as there are not so many that the function f becomes trivially hard for Arthur-Merlin protocols running in time n^c.

randomness tradeoffs for AM [SU09]. We refer to Theorem 2.27 in Section 2.4 for the full scaling and to Table 2.2 in the same section for other interesting instantiations.

Byproducts. Using our targeted hitting-set generators we are able to make progress on a number of related topics. We mention three representative ones here; more are described in Section 2.6.

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

Theorem 2.7. *If* $\mathsf{prAMTIME}[2^{\mathsf{polylog}(n)}] \subseteq \text{io-}\mathsf{NEXP}$*, then there exists a targeted hitting-set generator for* prAM *that yields the simulation* $\mathsf{prAM} \subseteq \text{io-}\mathsf{NTIME}[2^{n^c}]/n^\epsilon$ *for some constant* c *and all* $\epsilon > 0$*.*

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

Theorem 2.8. *If* $\mathsf{NTIME}[2^{an}] \cap \mathsf{coNTIME}[2^{an}] \not\subseteq \mathsf{BPTIME}[2^{(\log(a+1))^2 n}]^{\mathsf{SAT}}_{\|}$ *for some constant* $a > 0$*, then for every problem in* prAM *and all* $e > 0$ *there exists a simulation of the problem in* NP *that is correct on all but a fraction* $1/n^e$ *of the inputs of length* n *for infinitely many lengths* n*.*

The class $\mathsf{BPTIME}[t(n)]^{\mathsf{SAT}}_{\|}$ denotes probabilistic algorithms with bounded error that run in time $t(n)$ and can make parallel (i.e., non-adaptive) queries to an oracle for SAT. Theorem 2.8 answers a question in [GST03], which presents results in the different but

related "pseudo" setting, where the simulation may err on many inputs of any given length, but no polynomial-time nondeterministic algorithm can pinpoint an error at that length. We remark that our technique also leads to identical results in the "pseudo" setting by replacing the hardness assumption with hardness against $\mathsf{AMTIME}[t(n)]$.

The model $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ was used as a proxy for prAM in the initial derandomization results for Arthur-Merlin protocols [KvM02] and is seemingly more powerful. However, derandomization results for prAM typically translate into similar derandomization results for $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$. In particular, the conclusion $\mathsf{prAM} \subseteq \mathsf{NP}$ of Theorem 2.5 implies that $\mathsf{prBPP}_{\|}^{\mathsf{SAT}} \subseteq \mathsf{P}_{\|}^{\mathsf{SAT}}$, and the conclusion $\mathsf{prAM} \subseteq \mathsf{NTIME}[2^{n^c}]$ for some constant c in Theorem 2.6 implies that $\mathsf{prBPP}_{\|}^{\mathsf{SAT}} \subseteq \mathsf{DTIME}[2^{n^c}]_{\|}^{\mathsf{SAT}}$ for some constant c. In the case of Theorem 2.8, we argue that the hardness assumption implies simulations of $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ in $\mathsf{P}_{\|}^{\mathsf{SAT}}$ of the same strength as the simulations of prAM in NP. This way, we obtain a hardness vs. randomness tradeoff in which the hardness model and the model to-be-derandomized match, namely probabilistic algorithms with bounded error and non-adaptive access to an oracle for SAT.

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

Theorem 2.9. *For every problem in* prAM *and every constant* $\epsilon > 0$ *there exists a simulation of the problem in* $\Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$ *that is correct on all but a fraction* $1/n^e$ *of the inputs of length* n*, for all constants* e *and infinitely many lengths* n*.*

In fact, we can extend Theorem 2.9 to $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ in lieu of prAM. Similarly, we can extend the simulation of [Wil16] to $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ and thus also to prAM in lieu of AM.

Techniques. For our main result, we develop an instance-wise transformation of hardness into targeted hitting sets tailored for AM. In the setting of BPP, Chen and Tell combine the Nisan-Wigderson pseudorandom generator construction [NW94] with the doubly-efficient proof systems of Goldwasser, Kalai, and Rothblum [GKR15] (as simplified in [Gol18]). The latter allows them to capture the computation of a uniform circuit of size T and depth d for f on a given input x by a downward self-reducible sequence of polynomials, which they use to instantiate the NW generator. In case the derandomization of a one-sided error algorithm on a given input x fails, a bootstrapping strategy à la [IW01], based on a learning property of the NW generator, allows them to retrieve the value of $f(x)$ in time $O(d \cdot \mathsf{polylog}(T))$. Thus, provided the depth d is small compared to the size T, either the derandomization on input x works or else the computation of $f(x)$ can be sped up.

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

In order to remedy both shortcomings, we develop a new method to extract hardness from a nondeterministic computation on a given input x, based on probabilistically checkable proofs rather than [GKR15]. The soundness of our method presupposes some type of resilience of the underlying regular pseudorandom generator. The required property was first identified and used by Gutfreund, Shaltiel and Ta-Shma [GST03] for the Miltersen-Vinodchandran generator MV [MV05], and later by Shaltiel and Umans [SU09] for their recursive variant of the MV generator, RMV. We combine RMV with the probabilisti-

cally checkable proofs of Ben-Sasson, Goldreich, Harsha, Sudan, and Vadhan [BGH+06] to transform hardness into pseudorandomness for AM in an instance-wise fashion, without any uniform-circuit depth restriction or limitation on the amount of nondeterminism.

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

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

Organization. In Section 2.2, we develop the ideas behind our results and relate them to existing techniques. We start the formal treatment in Section 2.3 with definitions, notation, and other preliminaries. In Section 2.4, we construct our targeted hitting-set generator and establish our hardness-to-derandomization results that make use of it (Theorems 2.5 and 2.6). Section 2.5 presents the derandomization-to-hardness side of our near-equivalence, as well as a proof of our byproduct on derandomization to targeted hitting-set generators (Theorem 2.7). In Section 2.6, we derive our derandomization byproducts under uniform

worst-case hardness (the average-case simulation of Theorem 2.8 as well as a simulation that works on all inputs of infinitely many lengths). Section 2.7 contains our unconditional mild derandomization result for AM (Theorem 2.9).

2.2 Technical overview

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

2.2.1 Main results

We start with an overview of the techniques used for hardness-to-derandomization results in the traditional setting for BPP (lines 1 and 2 in Table 2.1), followed by those in the new setting (line 3 in Table 2.1). We then transition to AM, discuss the additional challenges, the known techniques in the traditional setting and, finally, our results in the new setting.

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

self-reducible or by its truth table being a codeword in a locally-correctable error-correcting code), then the exact function h can be reconstructed efficiently.

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

1. How to obtain the distinguishers D?

2. How to obtain the answers to the learning queries?

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

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

New setting for BPP. In the setting of line 3 in Table 2.1, the role of pseudorandom generators is taken over by *targeted* pseudorandom generators. Whereas PRGs are oblivious to x (beyond its length), targeted PRGs take x as an input and are only supposed to fool the randomized process on that particular x. This approach obviates the problem of obtaining the distinguisher D (question 1 above) as we can use $D = A(x, \cdot)$ for the given x. Targeted PRGs can be constructed from a PRG G by instantiating G with an oracle $h = h_x$ that

depends on x. This raises a third question in the application of a PRG for hardness vs. randomness tradeoffs:

3. How to obtain the function h_x from x?

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

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

Liu and Pass [LP23] also use the NW generator but obtain h_x as an encoding of 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 answers to the learning queries are provided as part of the information about $f(x)$ that is leaked, which allows them to reconstruct $f(x)$ directly and efficiently.

This approach leads to a (targeted) *pseudorandom* generator since it only involves a single instantiation of the NW generator. Reversing the hardness-to-derandomization direction yields an equivalence between derandomization of prBPP and the existence of almost-all inputs leakage-resilient hard functions.

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

Second, we need to accommodate *nondeterministic* algorithms computing the function f. This is because the direction from derandomization to hardness seems to need them (see Proposition 2.4). On each input x, such an algorithm needs to have at least one successful computation path, and on every successful computation path, the output should equal $f(x)$.

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

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

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

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

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

As for obtaining answers to the learning queries in the uniform setting for AM, we can make use of the nondeterminism allowed during the reconstruction and ask Merlin to provide the answers to the learning queries. However, we need to guard against a cheating Merlin. A strategy proposed by Gutfreund, Shaltiel and Ta-Shma in [GST03] consists of employing a function h that has a length-preserving instance checker. After Merlin has provided the supposed answers to the learning queries, to compute $h(z)$ for a given input z, we run the instance checker on input z and answer the queries y of the instance checker by running the

evaluator part of the reconstruction process on input y. All the runs of the evaluator can be executed in parallel, ensuring a bounded number of rounds overall, which can be reduced to two in the standard way at the cost of a polynomial blowup in the running time [BM88].

To guarantee soundness, the reconstruction process needs to have an additional *resilience* property, namely that it remains partial single-valued even when the learning queries are answered incorrectly. Two hitting-set generators tailored for AM are known to have the property: the Miltersen-Vinodchandran generator MV [MV05], which is geared toward the high end, and a recursive version, RMV, developed by Shaltiel and Umans [SU09] to cover a broader range. MV is used for the high end in [GST03], and RMV for the rest of the spectrum in [SU09].

New setting for AM. We build a targeted hitting-set generator for AM based on the RMV hitting-set generator. To obtain h_x from x, we make use of Probabilistically Checkable Proofs (PCPs) for the nondeterministic computation of the string $f(x)$ from x. Let V denote the verifier for such a PCP system that uses $O(\log(T(n))$ random bits and $\mathsf{polylog}(T(n))$ queries for nondeterministic computations that run in time $T(n)$. On input x, our targeted hitting-set generator 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 hitting-set generator 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 hitting-set generator fails.

For the reconstruction of the i-th bit of $f(x)$, Arthur generates the learning queries of the RMV reconstructor for the oracle y_i, and Merlin provides the purported answers as well as the value of the i-th bit of $f(x)$. Arthur then runs some random checks of the verifier V on input x, answering the verifier queries by executing the evaluator of the RMV reconstructor. All the executions of the evaluator can be performed in parallel, ensuring a bounded number

of rounds overall. The resilient partial single-valuedness property of the RMV reconstructor guarantees that the verifier queries are all consistent with some candidate proof $\tilde{y}_i$. The completeness and soundness of the PCP then imply the completeness and soundness of the reconstruction process for our targeted hitting-set generator. 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, R, which plays the role of a reconstructor for the targeted hitting-set generator. H and R have access to the input x and a co-nondeterministic circuit D, and have the following property.[3]

Property 2.10. *For every $x \in \{0,1\}^*$ and for every co-nondeterministic circuit D that accepts at least half of its inputs, at least one of the following holds:*

1. *$H(x, D)$ outputs a hitting set for D on every successful computation path.*
2. *$R(x, D)$ computes $f(x)$ in a complete and sound fashion.*

Theorem 2.5 follows by considering nondeterministic running time $T(n) = n^a$ and co-nondeterministic circuits D of size n^c for some $c > 1$. In this regime, H runs in time $n^{O(a+c)}$ and R in time $n^{O(c(\log a)^2)}$. Under the hypothesis of Theorem 2.5, the second item in Property 2.10 cannot happen except for finitely many x of length n, so the first item needs to hold. For any constant $c' < c$, this yields a polynomial-time targeted hitting-set generator for $\mathsf{prAMTIME}[n^{c'}]$, which can be used for all of prAM by padding. Theorem 2.6 follows along the same lines; the running time is dictated by the RMV reconstructor.

[3]The dependency of H on D is only through the number of input bits of D. For R, blackbox access to D suffices (in addition to the input x). However, we may as well give both H and R full access to the input x and the circuit D. In the intended application, the co-nondeterministic circuit D is obtained by hardwiring the input x into the Arthur-Merlin protocol being derandomized, but this is not essential for the construction.

We point out that the approach of Chen and Tell can be ported to the AM setting by replacing NW with a generator for AM that has the learning property and a reconstructor running in prAM. The nondeterminism allows us to run the bootstrapping process in parallel, so the number of rounds of Arthur and Merlin remains bounded, but the overall running time remains proportional to the depth of the circuits for f. This means that, like in the setting of BPP, this approach only yields meaningful results when the depth is small compared to the size. Nondeterministic circuits for f can be accommodated in this approach by treating them as deterministic circuits with nondeterministic guess bits as additional inputs. However, this limits the amount of nondeterminism that can be handled. Our approach based on PCPs remedies the limitations on depth as well as nondeterminism.

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

2.2.2 Byproducts

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

Targeted hitting-set generators from derandomization (Theorem 2.7). To obtain a targeted hitting-set generator from derandomization of prAM, we employ our targeted

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

Average-case derandomization (Theorem 2.8). Our average-case derandomization results under worst-case hardness assumptions also make use of our targeted hitting-set generator construction, but in a different way. They do not exploit the potential of the hitting sets to depend on the input x. In fact, they set $f(x)$ to the truth table of the worst-case hard language L from the hypothesis at an input length determined by $|x|$. Instead, they hinge on the strong resilient soundness properties of the reconstructor.

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

This approach necessitates a stronger form of resilience than the one provided by the RMV generator: That its reconstruction is sound when given as input *any* co-nondeterministic circuit D, not just those that accept at least half of their inputs (as in Property 2.10).

We don't know how to guarantee this with our prAM reconstruction, but we are able to do so in $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ by approximating the fraction of inputs that D accepts and outright failing if the fraction is too low.

We point out that earlier works [GST03; SU09] also manage to guarantee soundness of the reconstructor for co-nondeterministic circuits D that accept at least half of their inputs, based on the resilient partial single-valuedness of the reconstructor for MV or RMV. They do so by running an instance checker, which limits the hard function f to classes for which instance checkers are known to exist, such as complete problems for E and EXP. Instead, we achieve soundness of the reconstructor based on the soundness of a PCP. As PCPs exist for all nondeterministic computations, this makes our approach more suitable in this setting. In particular, we do not know how to obtain Theorem 2.8 along the lines of [GST03; SU09].

Unconditional mild derandomization (Theorem 2.9). Our unconditional mild derandomization result relies on a similar win-win argument as in the proof of Theorem 2.7: Either some hardness assumption/class separation holds, in which case we get derandomization right away, or we get a complexity collapse that we use to construct, by diagonalization, a hard function f that has the efficiency requirements we need to obtain the derandomization result using our targeted hitting-set generator.

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

2.3 Preliminaries

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

2.3.1 Nondeterministic, co-nondeterministic and single-valued computation

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

We are also interested in nondeterministic algorithms that compute total relations $R \subseteq \{0,1\}^* \times \{0,1\}^*$. 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 $\mathsf{NTIME}[T(n)]$.

2.3.2 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 samples a random string and sends it to Merlin. Merlin sends a string that depends on the input x and all prior communication from Arthur; the underlying function is referred to as Merlin's strategy, which is computationally unrestricted. At the end of the process, a deterministic computation on the input x and all communication determines acceptance. The running time of the process is the running time of the final deterministic computation.

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

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

Lemma 2.11 ([CR11]). $\mathsf{prBPP}_{\|}^{\mathsf{SAT}} \subseteq \mathsf{P}_{\|}^{\mathsf{prAM}}$.

In Lemma 2.11, the deterministic machines with oracle access to prAM on the right-hand side are guaranteed to work correctly irrespective of how the queries outside of the promise are answered, even if those queries are answered inconsistently, i.e., different answers may be given when the same query is made multiple times.

Arthur-Merlin protocols that output values. A promise Arthur-Merlin protocol P may also output a value. In this case, at the end of the interaction, the deterministic computation determines success/failure and, in case of success, an output value. We denote this value by $P(x, M)$, which is a random variable defined relative to a strategy M for Merlin. Similar to the setting of circuits, we indicate failure by setting $P(x, M) = \perp$, 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 2.12 (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: $(\forall M) \Pr[P(x, M) \notin \{v, \perp\}] \leq s$.
- P has partial single-valuedness s if there exists a value v such that P outputs v with soundness s. In symbols: $(\exists v)(\forall M) \Pr[P(x, M) \notin \{v, \perp\}] \leq s$.

◂

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

For a given function $f : X \to \{0,1\}^*$ where $X \subseteq \{0,1\}^*$, we say that P computes f with completeness $c(n)$ and soundness $s(n)$ if on every input $x \in X$, P outputs $f(x)$ with completeness $c(|x|)$ and soundness $s(|x|)$. Note that P may behave arbitrarily on inputs

that are not in X. In contrast, an AM protocol computing f still computes some value in a complete and sound fashion on inputs $x \notin X$.

2.3.3 Learn-and-evaluate and commit-and-evaluate protocols

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

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

Definition 2.13 (Learn-and-evaluate protocol). A learn-and-evaluate protocol P consists of a probabilistic oracle algorithm A_{learn} and a promise Arthur-Merlin protocol P_{eval}. Let $f : X \to \{0,1\}^*$ where $X \subseteq \{0,1\}^*$. We say that P computes f with error $e(n)$ for completeness $c(n)$ and soundness $s(n)$ if on every input $x \in X$ of length n the following hold: The probability over the randomness of A_{learn} that P_{eval} with input x and additional input $\pi = A^f_{\text{learn}}(1^n)$ outputs $f(x)$ with completeness $c(n)$ and soundness $s(n)$ is at least $1 - e(n)$. ◂

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

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

Definition 2.14 (Commit-and-evaluate protocol). A commit-and-evaluate protocol is a pair of promise Arthur-Merlin protocols $P = (P_{\text{commit}}, P_{\text{eval}})$. 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 (2.2) is at most $r(n)$:

$$\text{For every } x \text{ of length } n \text{ in } X,\ P_{\text{eval}}(x,\pi) \text{ has partial single-valuedness } s(n). \tag{2.2}$$

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

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

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

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).

2.3.4 Hitting-set generators and targeted hitting-set generators

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

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

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

Definition 2.15 (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 2.16 (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 $C(x,\cdot)$. ◂

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

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

Proposition 2.17. *There exists an algorithm that, on input 1^n and the description of a (Boolean output, two-round) $\mathsf{prAMTIME}[t(n)]$ protocol P, runs in time $O(t(n)^2)$ and outputs a co-nondeterministic circuit C of size $m = O(t(n)^2)$ that simulates and negates the computation of P for input length n, i.e., the input of C is comprised of $x \in \{0,1\}^n$ and Arthur's random-bit string r, and it co-nondeterministically verifies that there is no Merlin message that would lead to acceptance. In particular:*

- *If P with input x accepts all random inputs, then $D_x(r) \doteq C(x,r)$ rejects every input.*

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

2.3.5 PCPs and low-degree extensions

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

Lemma 2.18 ([BGH+05]). *Let T be a time bound. For every $s = s(n) : \mathbb{N} \to (0,1]$ and every language $L \in \mathsf{NTIME}[T(n)]$ there exists a PCP verifier V with perfect completeness, soundness s, randomness complexity $\log(1/s) \cdot (\log T(n) + O(\log\log T(n)))$, non-adaptive query complexity $\log(1/s) \cdot \mathsf{polylog}(T(n))$, and verification time $\log(1/s) \cdot \mathsf{poly}(n, \log T(n))$. More precisely,*

- *V has oracle access to a proof of length $T(n) \cdot \mathsf{polylog}(T(n))$, uses $\log(1/s) \cdot (\log T(n) + O(\log\log T(n)))$ random bits in any execution, makes $\log(1/s) \cdot \mathsf{polylog}(T(n))$ non-adaptive queries to the proof and runs in time $\log(1/s) \cdot \mathsf{poly}(n, \log T(n))$.*
- *If $x \in L$, $|x| = n$, then there exists y of length $T(n) \cdot \mathsf{polylog}(T(n))$ such that $\Pr[V^y(x) = 1] = 1$.*
- *If $x \notin L$, $|x| = n$, then for all y' of length $T(n) \cdot \mathsf{polylog}(T(n))$, $\Pr[V^{y'}(x) = 1] \leq s$.*

We also need standard 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 $n \cdot \mathsf{poly}(h, \log p, r) \leq \mathsf{poly}(h^r, \log p, r)$ given oracle access to x.

2.3.6 Average-case simulation

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

Definition 2.19 (Heuristic). Let Π be a promise-problem, $\mu : \mathbb{N} \to [0,1)$, $\mathcal{C}$ a complexity class and $\mathbf{x} = \{\mathbf{x}_n\}_{n\in\mathbb{N}}$ an ensemble of distributions where $\mathbf{x}_n$ is supported on $\{0,1\}^n$ and such that for all n, every x in the support of $\mathbf{x}_n$ satisfies the promise of Π. We write

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

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

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

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

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

2.4 Targeted hitting-set generator construction

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

Our construction builds on the RMV generator due to Shaltiel and Umans [SU09], which is a recursive variant of the MV generator that shares the desired resilience property with MV. We start with the definition of the RMV generator in Section 2.4.1 and state its reconstruction properties in terms of a commit-and-evaluate protocol. We present our construction and analysis in Section 2.4.2 and the derandomization consequences in Section 2.4.3.

2.4.1 Recursive Miltersen-Vinodchandran generator

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

Definition 2.20 (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 $\mathsf{poly}(s, t)$ 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 2.21 (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 2.21. It takes $(\Delta/r)^r \log p$ bits to describe these polynomials, but the local extractor only accesses $\mathsf{poly}(\Delta, r, \log p)$ 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.[4] Shaltiel and Umans call these MV lines, which we define next.

Definition 2.22 (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 2.21 for polynomials of degree Δ in $(r/2)$ variables over $\mathbb{F}$. Remember that $k = \Delta^5$ and that the extractor uses seed length $O(r \log p)$ and output length $t = \Delta \geq m$. By using only a prefix of the output, we have it output exactly m bits.

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

The construction runs in time $p^{O(r)}$ and therefore outputs at most that many strings. If the set output by the procedure fails as a hitting set for a co-nondeterministic circuit D of

[4] 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.

size m, then there exists an efficient commit-and-evaluate protocol P for $\hat{g}$ with additional input D. This is the main technical result of [SU09], which we present in a format that is suitable for obtaining our results.[5]

Lemma 2.23 ([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 $\log(1/s) \cdot \mathsf{poly}(\Delta, r)$ 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 $\log(1/s) \cdot \mathsf{poly}(\Delta, r)$ and Merlin replies with evaluations of $\hat{g}$ on each of the points in $S^2 \subseteq \mathbb{F}^r$. The 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.

[5] 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].

2.4.2 Targeted generator and reconstruction

In this section, we present our targeted hitting-set generator construction, which works as follows: On input x and a co-nondeterministic circuit D of size m, it guesses a PCP (as in Lemma 2.18) for each bit of $f(x)$ and verifies each PCP deterministically by enumerating over the PCP verifier's randomness. It encodes each PCP as a low-degree polynomial (as in Section 2.3.5), instantiates the RMV generator with each of the polynomials and outputs the union of the outputs for each instantiation. For the reconstruction, we have Merlin send a bit b and commit to the low-degree extension of a proof that the i-th bit of $f(x)$ equals b. Arthur then runs the PCP verifier using the evaluation protocol to answer proof queries. The protocol succeeds and outputs b if and only if the PCP verifier accepts. Here is the formal statement of the result.

Theorem 2.24. *Let $T(n)$ be a time bound and $f \in \mathsf{NTIME}[T(n)]$. There exists a nondeterministic algorithm H (the generator) that always has at least one successful computation path per input, and a promise Arthur-Merlin protocol R (the reconstructor) such that for every $x \in \{0,1\}^*$ and every co-nondeterministic circuit D that accepts at least half of its inputs, at least one of the following holds.*

1. *$H(x, D)$ outputs a hitting set for D on every successful computation path.*
2. *$R(x, D)$ computes $f(x)$ with completeness 1 and soundness 1/3.*

The construction also has the following properties:

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

- Input access: $H(x, D)$ *only depends on* x *and the size of* D*, and the only way* R *requires access to* D *is via blackbox access to the deterministic predicate that underlies* D*.*

Proof. Let $f \in \mathsf{NTIME}[T(n)]$, consider the language $L_f = \{(x, i, b) \mid f(x)_i = b\}$ and note that $L_f \in \mathsf{NTIME}[T(n)]$. Let V be the PCP verifier of Lemma 2.18 for L_f with soundness $s = s(n) = (100T(n)^{-1})$. Let also $h = h(m) = m^{100}$, $r = r(n, m)$ be the smallest power of 2 such that h^r is greater than the proof length of V on input length n and $p = p(n, m)$ be the smallest prime in the interval $[\Delta^{100}, 2\Delta^{100}]$ for $\Delta = h \cdot r$. Note, in particular, that $h^r = \mathsf{poly}(T(n), m)$ and $r = O(\log(T(n))/\log m)$.

Generator. The generator H, on input x and a co-nondeterministic circuit D of size m, first guesses the value of $z = f(x)$ and a proof y_i of the correct length $T(n) \cdot \mathsf{polylog}(T(n))$ for the i-th bit of z for each i. Then it verifies that $\Pr[V^{y_i}(x, i, z_i) = 1] = 1$ for all i by deterministically enumerating over the $\mathsf{poly}(T(n))$ random-bit strings for V. If any of the verifications fail, it fails. Otherwise, it outputs the union of $\mathrm{RMV}(\hat{y}_i)$ for all i, where $\hat{y}_i$ is the low-degree extension of y_i with parameters p, h and r. The initial verification step takes time $\mathsf{poly}(T(n))$, and executing $\mathrm{RMV}(\hat{y}_i)$ takes time $p^{O(r)} = \mathsf{poly}(T(n), m)$ and outputs strings of length m. This culminates in a running time of $\mathsf{poly}(T(n), m)$. Finally, since for the correct output $z = f(x)$ there always exist proofs y_i that are accepted with probability 1 for each i, there always exists a nondeterministic guess that leads the generator to succeed.

Reconstructor. We describe and analyze the prAM protocol R, which uses the commit-and-evaluate protocol $P = (P_{\text{commit}}, P_{\text{eval}})$ of Lemma 2.23 with soundness parameter $s' = s'(n) = (100T(n) \cdot q)^{-1}$, where $q = q(n) = \mathsf{polylog}(T(n))$ denotes the query complexity of V at input length n. On inputs x, D and an index i, Arthur and Merlin play the commit phase P_{commit}, which produces a commitment π_i to be fed into the evaluation phase. In parallel, Merlin also sends a bit b to Arthur. The idea is for an honest Merlin to send $b = f(x)_i$ and commit to the low-degree extension $\hat{y}_i$ of a proof y_i that witnesses $(x, i, b) \in L_f$ (or $f(x)_i = b$), though a

dishonest Merlin may send a different bit and/or commit to some different function. Let γ_i denote the function that Merlin committed to via P_{commit}, which may be accessed with high probability by executing the evaluation protocol P_{eval} with input π_i. The restriction of γ_i to $[h]^r$ defines a candidate PCP proof $\tilde{y}_i$. Arthur then runs the verifier $V^{\tilde{y}_i}(x, i, b)$, employing Merlin's help to evaluate $\tilde{y}_i$ whenever V makes a query to it (where binary queries are first converted into the respective $\vec{v} \in \mathbb{F}_p^r$ and all queries are evaluated in parallel). If $V^{\tilde{y}_i}(x, i, b)$ accepts, then R succeeds and outputs b, otherwise it fails.

Completeness. If D is not hit by $H(x, D)$, then for all indices i there exists at least one proof y_i that witnesses $(x, i, f(x)_i) \in L_f$ and such that $\text{RMV}(\hat{y}_i)$ fails to hit D, where $\hat{y}_i$ is the low-degree extension of y_i with parameters p, h and r. In that case, an honest Merlin can commit to such a $\hat{y}_i$ with probability 1 by the completeness property of Lemma 2.23 as well as send the correct value of $f(x)_i$ during the first phase. Then perfect completeness of V and P_{eval} guarantee that R succeeds and outputs $f(x)_i$ with probability 1.

(Resilient) soundness. If D accepts at least half of its inputs, then for a fixed index i the resilience property of P in Lemma 2.23 guarantees that with probability at least $1 - s'$, the commit phase is successful and thus the evaluation protocol with input π_i has partial single-valuedness s'. In that case, by a union bound over the at most q queries that V makes, with probability at least $1 - (100T(n))^{-1} = 1 - s$, every execution of the evaluation protocol results in the evaluation of a fixed function $\gamma_i : \mathbb{F}^r \to \mathbb{F}$. If Merlin sends the incorrect value of $b \neq f(x)_i$ in the first round (the only way he could try to have Arthur output the wrong value), the soundness property of V in Lemma 2.18 guarantees that R fails with probability at least $1 - s$ since $(x, i, b) \notin L_f$. By a union bound over these three "bad" events, all of which have probability at most s since $s \geq s'$, for any fixed index i, $R(x, D)$ with additional input i either fails or outputs $f(x)_i$ with probability at least $1 - 3s$. Finally, a union bound over the at most $T(n)$ possible indices i guarantees that R either fails or outputs $f(x)$ with soundness 1/3. In particular, if completeness also holds then $R(x, D)$ computes $f(x)$ with completeness 1 and soundness 1/3.

Efficiency. The commit phase takes time $\log(1/s') \cdot \mathsf{poly}(\Delta, r) = \mathsf{poly}(m, \log T(n))$ and two rounds of communication. Afterwards, evaluating each query made by $V(x, i, b)$ with P_{eval} takes time $(\log(1/s'))^2 \cdot \Delta^{O((\log r)^2)} = (m \cdot \mathsf{polylog}(T(n)))^{O((\log r)^2)}$. The verification step for V takes time $\log(1/s) \cdot \mathsf{poly}(n, \log T(n)) = \mathsf{poly}(n, \log T(n))$, and it makes at most $\log(1/s) \cdot \mathsf{polylog}(T(n)) = \mathsf{polylog}(T(n))$ queries, resulting in a total running time of $\mathsf{poly}(n) + (m \cdot \log T(n))^{O((\log r)^2)}$. Moreover, because V is non-adaptive, 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 protocol with two-rounds [BM88] leads to a prAM protocol with running time $\mathsf{poly}(n) \cdot (m \cdot \log T(n))^{O((\log r)^2)}$.

Input access. We observe that computing $\mathrm{RMV}(\hat{y}_i)$ for each i only requires knowledge of m, the size of circuit D (instead of the circuit itself) and thus the generator H also only requires knowledge of m. Similarly, the commit-and-evaluate protocol in Lemma 2.23 only requires blackbox access to the deterministic predicate that underlies the circuit D, and thus so does our reconstructor R since it just gives D as input to P. ∎

We remark that we can amplify the resilient soundness property for the reconstructor so that the probability that it outputs a value outside of $\{f(x)_i, \perp\}$ is at most 2^{-t} by running it $\Theta(t)$ times in parallel and outputting $\perp$ as soon as at least one of the answers is $\perp$ or the answers are inconsistent, and outputting the consistent answer bit otherwise.

We also present a version of the generator with a stronger resilient soundness property at the expense of increasing the complexity of the reconstructor from a promise Arthur-Merlin protocol to a probabilistic algorithm with parallel access to SAT. This version is useful for obtaining our byproducts in the average-case setting.

Corollary 2.25. *Let $T(n)$ be a time bound and $f \in \mathsf{NTIME}[T(n)]$. There exists a nondeterministic algorithm H (the generator) and a probabilistic algorithm R (the reconstructor) with parallel access to* SAT *that have the same properties as in Theorem 2.24 but such that the resilient soundness property holds for* every *co-nondeterministic circuit.*

In the setting of Corollary 2.25, item 2 of Theorem 2.24 should be interpreted as saying that $R(x, D)$ outputs $f(x)$ with probability at least 2/3. We refer to the stronger resilience property in Corollary 2.25 as *strong resilient soundness*.

The idea behind Corollary 2.25 is for the reconstructor to first check whether the co-nondeterministic circuit D accepts at least somewhat less than half of its inputs. This is where the parallel access to an oracle for SAT comes in; it allows us to distinguish with high probability between the cases where the fraction of accepted inputs is, say, at most 1/3 and at least 1/2. In the former case, the new reconstructor indicates failure with high probability. Otherwise, we boost the fraction of accepted inputs to at least 1/2 by trying D on two independent inputs, and then run the old reconstructor on the corresponding co-nondeterministic circuit D'.

Proof of Corollary 2.25. Let H' be the generator and R' the reconstructor of Theorem 2.24 instantiated with function f and amplified to have (resilient) soundness 1/6.

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

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

last two queries give inconsistent answers, then R declares failure. Otherwise, R outputs whatever R' does.

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

Correctness. If a co-nondeterministic circuit D accepts at least half of its inputs, so does the circuit D'. Moreover, if $H(x, D)$ fails to hit D, then $H'(x, D')$ fails to hit D'. The correctness of protocol R' then guarantees that there exists a strategy for Merlin that makes R' output $f(x)$ with probability 1, and no strategy can make R' output an incorrect value for $f(x)$ with probability at least $1/6$. It follows that the second parallel phase of R yields $f(x)$ with probability at least $5/6$. Accounting for the error probability of $1/6$ in the initial verification, we conclude that R outputs $f(x)$ with probability at least $2/3$.

Efficiency. The running time of H is asymptotically identical to that of H', and the running time of R is polynomial in the running time of R'.

Finally, the input access part follows right away from the input access part of Theorem 2.24. ■

Similar to the case of Theorem 2.24, we can amplify the strong resilient soundness property for the reconstructor so that the probability that it outputs a value outside of $\{f(x)_i, \perp\}$ (or different from $f(x)_i$ in case D is not hit by the generator) is at most 2^{-t} by running it $\Theta(t)$ times in parallel and outputting the majority answer.

2.4.3 Derandomization consequences

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

Theorem 2.26. *There exists a constant c such that the following holds. Let $t, T : \mathbb{N} \to \mathbb{N}$ be time bounds such that $t(n) \geq n$, $\Pi \in \mathsf{prAMTIME}[t(n)]$ and $\{\mathbf{x}_n\}_{n\in\mathbb{N}}$ be an ensemble of distributions such that $\mathbf{x}_n$ is supported over $\{0,1\}^n$ and such that for all n, every x in the support of $\mathbf{x}_n$ satisfies the promise of Π. Assume that for $\mu : \mathbb{N} \to [0,1)$ there exists a length-preserving function $f \in \mathsf{NTIME}[T(n)]$ such that for every $\mathsf{prAMTIME}[t(n)^{O((\log r)^2)}]$ protocol P for $r = O(\log(T(n))/\log(t(n)))$, it holds that the probability over $x \sim \mathbf{x}_n$ that $P(x) = f(x)$ is at most $\mu(n)$ for all but finitely many n. Then, it holds that*

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

Proof. First, notice that if $t(n) \leq \log T(n)$, then the conclusion is trivial and if $t(n) \geq T(n)$ then the premise is impossible, so we focus on the case that $\log T(n) \leq t(n) \leq T(n)$. Let $\Pi \in \mathsf{prAMTIME}[t(n)]$ and let P be a two-round protocol for Π running in time $O(t(n))$ on inputs of length n. On input $x \in \{0,1\}^n$, compute the circuit D_x of Proposition 2.17 with protocol P, and note that D_x has size $O(t(n)^2)$. Then, instantiate the hitting-set generator of Theorem 2.24 with f. Feed H inputs x and D_x and run the usual derandomization procedure for protocol P with the set output by $H(x, D_x)$: For each string $\rho \in H(x, D_x)$, nondeterministically guess Merlin's message y_ρ and compute the output of P with randomness ρ and message y_ρ, accepting if and only if P accepts for every $\rho \in H(x, D_x)$. The entire procedure runs in nondeterministic time $\mathsf{poly}(T(n), t(n)) = O(T(n)^c)$ for some constant c, since $T(n) \geq t(n)$.

Assume, with the intent of deriving a contradiction, that with probability at least $\mu(n)$ over $x \sim \mathbf{x}_n$, this derandomization fails for input x. First, notice that by the perfect completeness of P it must be the case that such an x lies in Π_N and that P with input x accepts every string in $H(x, D_x)$. Therefore, D_x acts as a distinguisher for $H(x, D_x)$, i.e., it rejects

every string output by D_x while accepting at least half of its inputs. By computing D_x and feeding it to the prAM protocol R of Theorem 2.24, we obtain a prAM protocol that computes individual bits of $f(x)$ correctly for every x for which the derandomization fails, i.e., with probability at least $\mu(n)$ over $x \sim \mathbf{x}_n$. By running this protocol n times in parallel to compute every bit of $f(x)$, we obtain a prAM protocol that runs in time

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

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

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

As a consequence of Theorem 2.26, if the hardness assumption holds for almost-all inputs, then we obtain full derandomization of prAM.

Theorem 2.27. *There exists a constant c such that the following holds. Let $t, T : \mathbb{N} \to \mathbb{N}$ be time bounds such that $t(n) \geq n$. If there is a length-preserving function $f \in \mathsf{NTIME}[T(n)]$ that is hard on almost-all inputs against $\mathsf{prAMTIME}[t(n)^{O((\log r)^2)}]$ for $r = O(\log(T(n))/\log(t(n)))$ then*

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

Moreover, there exists a targeted hitting-set generator that achieves this derandomization result.

Proof. The statement follows from Theorem 2.26 by noting that the assumption that f is hard on almost-all inputs implies that f is hard for all possible distributions $\mathbf{x}_n$ with success probability $\mu(n) = 0$. In particular, the following nondeterministic algorithm is a hitting-set

generator for prAM: On input $x \in \{0,1\}^*$ and a co-nondeterministic circuit C of size m, output $H(x, D)$ where H is the generator of Theorem 2.24 and $D \doteq C(x, \cdot)$. This algorithm has a successful computation path for any input and, on every successful computation path on inputs where D accepts at least half of its inputs, it outputs a set that hits D. The running time of the generator is $\mathsf{poly}(T(n), m)$. ■

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

Table 2.2: Derandomization consequences that follow from different instantiations of Theorem 2.27.

By setting parameters in Theorem 2.27, we obtain the derandomization results listed on Table 2.2. In particular, the first line of Table 2.2 establishes Theorem 2.5 and the last line establishes Theorem 2.6. We now provide more details on how to obtain each line of Table 2.2:

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

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

- For the low end, let $\nu = \nu(n) = o(1)$ be such that $T(n) = 2^{n^\nu}$ and set $t(n) = n$. In this case, $r \leq n^\nu$. Then, $\mathsf{prAMTIME}[n] \subseteq \mathsf{NTIME}[\mathsf{poly}(n, 2^{n^\nu})]$ follows as long as f is hard on almost-all inputs against $\mathsf{prAMTIME}[n^{O((\nu \log n)^2)}]$. Since $\mathsf{poly}(n, 2^{n^\nu}) = 2^{n^{o(1)}}$ and $n^{O((\nu \log n)^2)} = n^{o((\log n)^2)}$, the result for prAM follows by padding.

- For the very low end, set $t(n) = n^b$ for a constant b, in which case $r = \mathsf{poly}(n)$. Then, $\mathsf{prAMTIME}[n^b] \subseteq \mathsf{NTIME}[2^{n^c}]$ for some constant c follows as long as f is hard on almost-all inputs against $\mathsf{prAMTIME}[n^{O(b(\log n)^2)}]$. To get the result for prAM, it suffices for hardness to hold for all b.

2.5 Consequences of derandomization

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

2.5.1 Hardness on almost-all inputs

We start with our derandomization-to-hardness implication: If $\mathsf{prAM} \subseteq \mathsf{NP}$ then for all constants c there is a length-preserving function f computable in nondeterministic polynomial time (with a few bits of advice) that is hard on almost-all inputs against $\mathsf{AMTIME}[n^c]$. The basic idea is that, under the derandomization hypothesis, every (single-bit) AM protocol that runs in time n^c can be simulated by a single-valued nondeterministic machine without too much time overhead. If we have as advice whether a particular nondeterministic machine is single-valued or not at input length n, we can negate its input efficiently, obtaining a function f computable in nondeterministic time $\mathsf{poly}(n)$ that is almost-all inputs hard against AM protocols that run in time n^c. We now state Proposition 2.4 formally.

Proposition 2.28 (Formal version of Proposition 2.4). *If* $\mathsf{prAM} \subseteq \mathsf{NP}$*, then for every constant* c *and increasing function* $\alpha : \mathbb{N} \to \mathbb{N}$ *there exists a length-preserving function* $f \in \mathsf{NP}/\alpha(n)$ *that is hard on almost-all inputs against* $\mathsf{AMTIME}[n^c]$*.*

Proof. Assume that $\mathsf{prAM} \subseteq \mathsf{NP}$ and let c' be a constant to be defined later (which depends on c). The basic idea for the function f is as follows: On an input x of length n, we set its i-th output bit (for $1 \leq i \leq \min(n, \alpha(n))$) to the opposite of the i-th bit output by the i-th

nondeterministic Turing machine N_i on input x (if N_i is single-valued and halts in at most $n^{c'+2}$ steps at input length n), and otherwise we set it to 0. Formally, on input x of length n and for $1 \leq i \leq n$

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

Note that f is computable by a single-valued nondeterministic machine running in time $O(n^{c'+3})$ with $\alpha(n)$ bits of advice (indicating whether N_i is single-valued and halts in at most $n^{c'+2}$ steps at input length n for $1 \leq i \leq \alpha(n)$).[6] This holds because, when N_i is single-valued, computing $1 - N_i(x)_i$ can be done by guessing a path on which N_i succeeds, which must result in the unique value $N_i(x)$, and then outputting the opposite of the i-th bit of that. Assume, with the intent of deriving a contradiction, that there exists an AM protocol P that runs in time $O(n^c)$ and computes f on an infinite set of inputs $X \subseteq \{0,1\}^*$. Consider the protocol P' that takes as regular input a triple (x, i, b) and accepts iff the i-th bit of the output of protocol P with input x equals b (if $i > |x|$ then P' rejects). Note that P' induces a language L in $\mathsf{AMTIME}[n^c]$. Since $\mathsf{prAM} \subseteq \mathsf{NP}$ and $\mathsf{prAMTIME}[n^c]$ has a complete problem under linear-time reductions, it follows that there exists a constant c' such that $\mathsf{AMTIME}[n^c] \subseteq \mathsf{NTIME}[n^{c'}]$.[7] Let N be a nondeterministic machine that runs in time $n^{c'}$ and computes L. Note that for every $x \in \{0,1\}^*$ and $1 \leq i \leq |x|$, $N(x, i, b) = 1$ for exactly one $b \in \{0,1\}$, and when $x \in X$, $N(x, i, b) = 1$ if and only if $f(x)_i = b$.

Now consider the following procedure N': On input $x \in \{0,1\}^n$, guess a value b_i and a witness y_i for each $1 \leq i \leq n$ and run $N(x, i, b_i; y_i)$. If for all i, $N(x, i, b_i; y_i)$ accepts, N' succeeds and prints the concatenation of the guessed b_i's, otherwise N' fails. Note that N' is a nondeterministic machine that runs in time $O(n^{c'+1})$. Moreover, by our assumption that P

[6] The nondeterministic machine computing f is only guaranteed to be single-valued when given the correct advice string.

[7] While our argument only requires that there exists a constant c' such that $\mathsf{AMTIME}[n^c] \subseteq \mathsf{NTIME}[n^{c'}]$, we use the assumption $\mathsf{prAM} \subseteq \mathsf{NP}$ instead of $\mathsf{AM} \subseteq \mathsf{NP}$ since it is unknown whether $\mathsf{AMTIME}[n^c]$ contains a complete problem under linear-time reductions.

is an AM protocol and that $\mathsf{prAM} \subseteq \mathsf{NP}$, N' is single-valued on *every* input. By construction, the single value equals $f(x)$ for all $x \in X$.

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

This result extends to other parameter settings. As an example, we state a version of Proposition 2.28 at the very low end.

Proposition 2.29. *If there exists a constant c such that that $\mathsf{AM} \subseteq \mathsf{NTIME}[2^{n^c}]$, then for every increasing function $\alpha : \mathbb{N} \to \mathbb{N}$ there exists a function $f \in \mathsf{NEXP}/\alpha(n)$ that is hard on almost-all inputs against AM protocols running in polynomial time.*

Proof (Sketch). The proof is essentially identical to that of Proposition 2.28, but with a different time bound. Since $\mathsf{AM} \subseteq \mathsf{NTIME}[2^{n^c}]$, the diagonalizing machine N needs to diagonalize against single-valued nondeterministic algorithms running in time $2^{n^{c'}}$ for some fixed constant $c' > c$, and thus we get a nondeterministic algorithm that runs in time $O(2^{n^k})$ for any constant $k > c'$. ∎

We conclude this section by noting in more detail where the gaps between our hardness-to-derandomization and derandomization-to-hardness results lie. The first gap lies in the fact that in the derandomization-to-hardness direction, the hard function f we construct requires a few bits of advice that we don't know how to handle in the other direction. There is, however, a subtler difference — In the hardness-to-derandomization direction, we require hardness against prAM protocols, which may not obey the AM promise on all inputs (though we only consider the protocol as computing $f(x)$ on input x if it obeys the promise and respects both completeness and soundness on input x). In the derandomization-to-hardness direction, we can only guarantee hardness against AM protocols, which necessarily obey the AM promise on all inputs. We remark that a similar problem shows up in other

hardness vs. randomness tradeoffs for AM [GST03; SU09]. For example, to conclude almost-everywhere derandomization of AM, the authors of [GST03] require hardness of EXP against AM protocols for which completeness only holds infinitely-often. Finally, we also note that, while Chen and Tell only state their derandomization-to-hardness result for BPP [CT21], in that setting one can actually achieve hardness against prBPP (where the probabilistic algorithm might not have a high-probability output for every input).

2.5.2 Targeted hitting-set generator

In this section, we prove Theorem 2.7 along the lines of the intuition provided in Section 2.2.2. We make use of a win-win argument: Either the $\mathsf{EXP} \neq \mathsf{NEXP}$ hardness assumption holds, in which case there is a regular (oblivious) hitting-set generator that guarantees the derandomization result [IKW02]. Or else we may assume that $\mathsf{EXP} = \mathsf{NEXP}$, which allows us to construct a function f that is hard against prAM protocols by diagonalization, with which we then instantiate Theorem 2.24 to obtain the targeted hitting-set generator.

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

Lemma 2.30 ([IKW02]). *If* $\mathsf{NEXP} \neq \mathsf{EXP}$ *then* $\mathsf{prAM} \subseteq \text{io-}\mathsf{NTIME}[2^{n^\epsilon}]/n^\epsilon$ *for every* $\epsilon > 0$*. Moreover, for every* $\epsilon > 0$ *there exists a (regular) hitting-set generator that achieves this derandomization.*

We now prove Theorem 2.7, which we restate here for convenience.

Theorem 2.31 (Theorem 2.7, restated). *If* $\mathsf{prAMTIME}[2^{\mathsf{polylog}(n)}] \subseteq \text{io-}\mathsf{NEXP}$*, then for some constant* c *and all* $\epsilon > 0$*, there exists a targeted hitting-set generator for* prAM *that yields the simulation* $\mathsf{prAM} \subseteq \text{io-}\mathsf{NTIME}[2^{n^c}]/n^\epsilon$*.*

Proof. If $\mathsf{EXP} \neq \mathsf{NEXP}$, we are done by Lemma 2.30. Otherwise, it holds that $\mathsf{NEXP} = \mathsf{EXP}$. We use this collapse to construct a length-preserving multi-bit function $f \in \mathsf{EXP}$ that is hard against $\mathsf{prAMTIME}[n^{(\log n)^3}]$. We then instantiate Theorem 2.24 with f to obtain the targeted

hitting-set generator. Hardness against protocols running in this time bound suffices along the lines of Theorem 2.6.

Before constructing f, we make an observation: Due to the instance-wise nature of our construction, to obtain an infinitely-often derandomization result using Theorem 2.24 it suffices to have an *infinitely-often all-inputs hardness assumption*. More precisely, we require the following: For every $\mathsf{prAMTIME}[n^{(\log n)^3}]$ protocol P, there exist infinitely many input lengths n such that P fails to compute f for every x of length n. Thus, we construct a function f with this requirement in mind.

Under the hypothesized derandomization assumption and because $\mathsf{prAMTIME}[n^{(\log n)^3}]$ has a complete problem under linear-time reductions, it follows that there exists a constant k such that $\mathsf{prAMTIME}[n^{(\log n)^3}] \subseteq \text{io-}\mathsf{NTIME}[2^{n^k}]$. Since $\mathsf{NTIME}[2^{n^k}]$ also has a complete problem under linear-time reductions, under the assumption $\mathsf{EXP} = \mathsf{NEXP}$, there exists a constant k' such that $\mathsf{prAMTIME}[n^{(\log n)^3}] \subseteq \text{io-}\mathsf{DTIME}[2^{n^{k'}}]$. In that case, it suffices to diagonalize against fixed-exponential time machines to construct f. Similar to Proposition 2.28, we define the i-th bit of $f(x)$ to be the opposite of the i-th bit output by $M_i(x)$ when it runs for at most $2^{|x|^{k'}+1}$ steps, where M_i is the i-th deterministic Turing machine. Formally, on input x of length n and for $1 \leq i \leq n$,

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

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

Assume, toward a contradiction, that there exists a $\mathsf{prAMTIME}[n^{(\log n)^3}]$ protocol P such that for almost-all input lengths n, P computes f on at least one input $x \in \{0,1\}^n$, and call the set of inputs where P computes f correctly X. Again, similar to the proof of Proposition 2.28, P induces a problem Π in $\mathsf{prAMTIME}[n^{(\log n)^3}]$, and by our assumptions, there is a language $L \in \mathsf{DTIME}[2^{n^{k'}}]$ such that L and Π agree on infinitely many input lengths. Let M be a deterministic Turing machine running in time $O(2^{n^{k'}})$ that decides

L. Recall that yes-instances of Π are triples (x, i, b) such that $x \in X$ and $f(x)_i = b$ while no-instances have $x \in X$ and $f(x)_i \neq b$. Let M' be the deterministic Turing machine that, on input x of length n, outputs $M'(x)$ of length n such that $M'(x)_i = 1$ if and only if M accepts $(x, i, 1)$ for $1 \leq i \leq n$. Note that M' runs in time $2^{n^{k'+1}}$. By construction and our assumption on P, for infinitely many input lengths n there exists at least one $x \in X \cap \{0,1\}^n$ such that $M'(x) = f(x)$.

Let i be the index of M' in our enumeration. By definition of f, for every input $x \in \{0,1\}^*$ of sufficiently large length $n \geq i$ (so that it has a chance to negate the output of M'), and in particular for all sufficiently large inputs $x \in X$, we have that $f(x)_i = 1 - M'(x)_i = 1 - f(x)_i$, a contradiction. Finally, we instantiate Theorem 2.26 with f to obtain a targeted hitting-set generator for prAM that runs in exponential time, which suffices to obtain the conclusion. ■

2.6 Derandomization under uniform worst-case hardness

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

2.6.1 Average-case simulation

In this section, we develop our average-case derandomization results for prAM under worst-case uniform hardness assumptions (where hardness is against $\mathsf{BPTIME}_{\|}^{\mathsf{SAT}}$). Our results in this setting work as follows: Assume there exists a hard language $L \in \mathsf{NTIME}[T(n)] \cap \mathsf{coNTIME}[T(n)]$. To derandomize some prAM protocol P on input length n, we first consider

the hard language L at some suitable input length ℓ, which depends on the hardness of L (for Theorem 2.8, for example, we take $\ell = \Theta(\log n)$). Then we let f be the function that maps any input $x \in \{0,1\}^n$ to the truth table of L at input length ℓ, and it follows from the complexity of L that $f \in \mathsf{NTIME}[2^\ell \cdot T(\ell)]$. Finally, we instantiate our targeted hitting-set generator construction H with f and use it to derandomize P.

For the reconstruction, we make use of the strong resilient soundness property of Corollary 2.25. If the average-case derandomization fails, to decide whether z of length ℓ is in L, we first sample multiple candidate "good" strings x that hopefully lead to a distinguisher D_x for the generator (enough so that we expect at least one "good" x with high probability). Then, we run the reconstruction for all of them, accepting if and only if at least one of those outputs 1. By the strong resilient soundness property and amplification, with high probability every execution either fails or outputs $f(x)_z = L(z)$, and in the high probability case that we sample at least one "good" x, some execution outputs $L(z)$, meaning we can compute L efficiently on input length ℓ.

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

Theorem 2.32 (Strengthening of Theorem 2.8). *If* $\mathsf{NTIME}[2^{an}] \cap \mathsf{coNTIME}[2^{an}]$ *is not included in* $\mathsf{BPTIME}[2^{(\log(a+1))^2 n}]^{\mathsf{SAT}}_{\|}$ *for some constant* $a > 0$*, then for all* $e > 0$ *it holds that*

$$\mathsf{prAM} \subseteq \text{io-}\mathsf{Heur}_{1/n^e}\mathsf{NP}$$

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

Proof. We first argue the result for prAM. Consider derandomizing a prAM protocol P for a problem Π running in time $O(n^k)$ for some constant k. Let S be an $O(n^s)$-time sampler for a distribution in $\{0,1\}^n$ and e be a constant such that we want to "fool" S with probability at least $1 - 1/n^e$. Let f be a function mapping every $x \in \{0,1\}^n$ to the truth table of the hard language $L \in \mathsf{NTIME}[2^{an}] \cap \mathsf{coNTIME}[2^{an}]$ at input length $\ell = \ell(n) = \Theta(\log n)$ to be set precisely later. Note that $f \in \mathsf{NTIME}[T(n)]$ for $T(n) = 2^{(a+1)\ell}$. Instantiate the generator H of Corollary 2.25 with f, run H on input $x = 0^n$ (recall f maps every string in $\{0,1\}^n$ to the

same truth table) and co-nondeterministic circuit size $m = O(n^{2k})$, and use it to attempt to derandomize P in nondeterministic time $\mathsf{poly}(T(n), n^{2k}) = \mathsf{poly}(n)$.

If the derandomization fails for almost-all input lengths, even heuristically, then for almost-all input lengths n, $S(1^n)$ outputs with probability at least $1/n^e$ a string $x \in \{0,1\}^n$ such that the simulation errs on x, i.e., the circuit D_x obtained from x and P using Proposition 2.17 is a distinguisher for $H(0^n, D_x)$. To compute L at input length ℓ, it then suffices to do the following: On input $z \in \{0,1\}^\ell$, first use S to sample $t = \Theta(n^e)$ inputs $x_1, \ldots, x_t$ and use these to construct a list $D_{x_1}, \ldots, D_{x_t}$ of candidate distinguishers for $H(0^n, D_x)$. With high probability, this list contains an actual distinguisher for the generator. Let R be the algorithm of Corollary 2.25, amplified by parallel repetition to have negligible soundness 2^{-n}, i.e., with probability at least $1 - 2^n$, the algorithm outputs either $f(x)$ or $\perp$. Finally, run R with inputs 0^n, index z (recall $f(0^n)$ equals the truth table of L at input length ℓ) and D_{x_i} for every sampled input x_i, and accept if and only if some execution outputs 1. To see that this is correct, note that by a union bound, with high probability every execution of R is successful in the sense that it either outputs $f(0^n)_z = L(z)$ or $\perp$. Conditioned on there being a distinguisher in the list, we are guaranteed to output the correct value of $L(z)$ with high probability.

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

Now, we argue the result for $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$. To do so, we use the containment $\mathsf{prBPP}_{\|}^{\mathsf{SAT}} \subseteq \mathsf{P}_{\|}^{\mathsf{prAM}}$ [CR11]. It suffices to show that every deterministic polynomial-time algorithm with non-adaptive oracle access to a paddable prAM-complete problem $\Gamma \in \mathsf{prAMTIME}[n]$ can be simulated by deterministic polynomial-time algorithms with non-adaptive oracle access to

SAT. Let M be a deterministic algorithm with non-adaptive oracle access to Γ running in time $O(n^b)$ and S be an $O(n^s)$-time sampler that we want to "fool" with probability at least $1 - 1/n^e$. Since Γ is paddable, we may assume that every query made by M on inputs of length n is of length $O(n^b)$ (at the expense of increasing its running time to $O(n^{2b})$). To simulate M on input x, let f be a function mapping every $x \in \{0,1\}^n$ to the truth table of L at input length $\ell = \ell(n) = \Theta(\log n)$. As before, $f \in \mathsf{NTIME}[2^{(a+1)\ell}]$. Instantiate the generator H of Corollary 2.25 with f and use it to derandomize Γ at input length $O(n^b)$ in order to obtain a $\mathsf{P}_{\|}^{\mathsf{SAT}}$ simulation for M. Whenever M with input x queries Γ, we instead query the SAT oracle whether the nondeterministic simulation of Γ using H with input 0^n and co-nondeterministic circuit size $m = O(n^{2b})$ accepts. This simulation runs in $\mathsf{P}_{\|}^{\mathsf{SAT}}$ since M is non-adaptive.

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

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

Theorem 2.33. *If* $\mathsf{NEXP} \cap \mathsf{coNEXP} \not\subseteq \mathsf{BPTIME}[n^{b((\log n)^2)}]_{\|}^{\mathsf{SAT}}$ *for all* $b > 0$*, then for every*

$\epsilon > 0$ *and all* $e > 0$

$$\mathsf{prAM} \subseteq \text{io-}\mathsf{Heur}_{1/n^e}\mathsf{NTIME}[2^{n^\epsilon}]$$

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

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

Proof. We begin with the argument for prAM. Let L be a hard language in $\mathsf{NTIME}[2^{n^a}] \cap \mathsf{coNTIME}[2^{n^a}]$ for some constant $a \geq 1$. Consider derandomizing a protocol P for a problem $\Pi \in \mathsf{prAMTIME}[n^k]$ for constant k. Let $\epsilon > 0$ and f be the function mapping every $x \in \{0,1\}^n$ to the truth table of L at input length $\ell = n^\epsilon$. Note that $f \in \mathsf{NTIME}[T(n)]$ for $T(n) = 2^{n^{a\epsilon}}$. Instantiate the generator H of Corollary 2.25 with f, run H on input $x = 0^n$ and co-nondeterministic circuit size $m = O(n^{2k})$, and use it to derandomize P. The simulation runs in nondeterministic time $\mathsf{poly}(T(n), n^{2k})$, which is at most $2^{n^{\epsilon'}}$ for any $\epsilon' > 0$ by taking a sufficiently small $\epsilon > 0$.

The reconstruction is identical to that of Theorem 2.32 but with $\ell = n^\epsilon$. The running time is $O(n^{e+s})$ to generate the samples and $(n^{2k})^{O((\log r)^2)}$ per sample for running R, where $r = O(\log(T(n))/\log n)$, for a total of $O(n^e(n^s + n^{O((\log r)^2)}))$. Given our parameter choices, $r = O(n^{a\epsilon})$, and the expression is upper bounded by $O(n^e(n^s + n^{O((a\epsilon \log n)^2)}))$. As the input length is $\ell = n^\epsilon$ for constant ϵ, there exists a constant b (depending on a, e, s, ϵ) such that the running time is upper bounded by $\ell^{b(\log n)^2}$. If hardness holds for all $b > 0$, then the same simulation works for any constant value of s and e, i.e., for any polynomial-time sampler and any inverse-polynomial error probability.

The proof for $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ is also almost identical to that of Theorem 2.32, where we derandomize the "oracle" Γ using the generator H from Corollary 2.25 instantiated with the function f that maps every $x \in \{0,1\}^n$ to the truth table of L at input length $\ell = n^\epsilon$ and use a set of queries instead of a set of inputs to obtain the list of candidate distinguishers for the reconstruction. This approach naturally leads to a simulation in $\mathsf{P}_{\|}^{\mathsf{NTIME}[2^{n^\epsilon}]}$, and

we obtain the $\mathsf{DTIME}[2^{n^\epsilon}]^{\mathsf{SAT}}_{\|}$ simulation by replacing the original queries with padded SAT queries. ■

2.6.2 Infinitely-often all-input simulation

By introducing nondeterminism in the algorithms we require hardness for, we are able to extend Theorem 2.8 to conclude full (infinitely-often) derandomization of prAM. We have shown that, if the hitting-set generator construction of Theorem 2.8 fails to obtain average-case derandomization of prAM, then we are able to efficiently sample candidate distinguishers with the hope that at least one is "good". However, if the hitting-set generator fails in the worst case, it is harder to pinpoint exactly where it does so as to obtain a distinguisher. To solve this, we have Merlin send a "good" input x. This necessitates a lower bound against $\mathsf{MATIME}^{\mathsf{SAT}}_{\|}$, but allows for concluding full (infinitely-often) derandomization of prAM and $\mathsf{prBPP}^{\mathsf{SAT}}_{\|}$.

Theorem 2.34. *If* $\mathsf{NTIME}[2^{an}] \cap \mathsf{coNTIME}[2^{an}] \notin \mathsf{MATIME}[2^{(\log(a+1))^2\ell)}]^{\mathsf{SAT}}_{\|}$ *for some constant* $a > 0$*, then*

$$\mathsf{prAM} \subseteq \text{io-}\mathsf{NP}$$

$$\mathsf{prBPP}^{\mathsf{SAT}}_{\|} \subseteq \text{io-}\mathsf{P}^{\mathsf{SAT}}_{\|}.$$

Proof. We argue the result for prAM first. Let $\Pi \in \mathsf{prAMTIME}[n^k]$ for some constant k and let L be a hard language in $\mathsf{NTIME}[2^{an}] \cap \mathsf{coNTIME}[2^{an}]$. Let f be a function mapping every string in $\{0,1\}^n$ to the truth table of L at input length $\ell = \Theta(\log n)$ to be set precisely later. Note that $f \in \mathsf{NTIME}[T(n)]$ for $T(n) = 2^{(a+1)\ell}$. Instantiate the generator H of Corollary 2.25 with f, run H on input 0^n and co-nondeterministic circuit size $m = O(n^{2k})$, and use it to derandomize P in time $\mathsf{poly}(T(n), n) = \mathsf{poly}(n)$.

If the simulation fails for some input of almost-all input lengths, then for almost-all input lengths n there exists an $x \in \Pi_N$ of length n such that the simulation errs on x, i.e., the circuit D_x of Proposition 2.17 instantiated with the protocol for Π and x is a distinguisher

for $H(0^n, D_x)$. Let R be the reconstructor of Corollary 2.25 and consider the following Merlin-Arthur protocol for L, where the protocol has parallel oracle access to SAT: On input $z \in \{0,1\}^\ell$, Merlin sends x, and Arthur runs $R(0^n, D_x)$ to compute the z-th bit of $f(0^n)$ (which equals $L(z)$). If R outputs $\perp$, then the protocol rejects, otherwise, it accepts if and only if R outputs 1. Because R is a probabilistic algorithm with parallel access to an oracle for SAT, Arthur can sample the randomness required for it and then run the underlying deterministic parallel-SAT-oracle computation, meaning this is indeed a $\mathsf{MA}_{\parallel}^{\mathsf{SAT}}$ protocol. Completeness follows since Merlin can send a correct value of x, and soundness follows from the strong resilience property of R: Even if Merlin sends a "bad" x', R is still guaranteed to either fail or output $L(z)$ with high probability.

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

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

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

some $\epsilon > 0$ [Lu01].

2.7 Unconditional mild derandomization

In this section, we establish our unconditional mild derandomization result for prAM and extend it to $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$. We employ a similar win-win argument to that of the proof of Theorem 2.7: Either some hardness assumption/class separation holds (here, $\Sigma_2\mathsf{EXP} \not\subseteq \mathsf{NP/poly}$), in which case we get derandomization right away. Or else we get a complexity collapse which we can use to construct a hard function f that has the efficiency requirements we need to apply one of our targeted hitting-set constructions (in this case Theorem 2.33, which requires hardness against $\mathsf{BPTIME}[2^{\mathsf{polylog}(n)}]_{\|}^{\mathsf{SAT}}$).

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

- $\Sigma_2\mathsf{EXP} \not\subseteq \mathsf{NP/poly}$. In this case, the derandomization result follows from standard hardness vs. randomness tradeoffs.
- $\Sigma_2\mathsf{EXP} \subseteq \mathsf{NP/poly}$. In this case, we diagonalize against $\mathsf{BPTIME}[2^{\mathsf{polylog}(n)}]_{\|}^{\mathsf{SAT}}$ in $\mathsf{P}^{\Sigma_2\mathsf{EXP}} = \mathsf{EXP}$, and then instantiate Theorem 2.33 to conclude the proof.

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

2.7.1 Nondeterministic easy witnesses

In this section, we prove our "easy witness lemma" for $\Sigma_2\mathsf{EXP}$. One way of thinking of Σ_2 computations is as follows: On input x, guess a string y and then run a co-nondeterministic verifier on input (x, y). This view allows us to abstract the co-nondeterministic verification

and think of y as a witness for x. In this section, we show that if $\Sigma_2\mathsf{EXP} \subseteq \mathsf{NP/poly}$, then every language in $\Sigma_2\mathsf{EXP}$ has witnesses that are the truth tables of functions computed by polynomial-size single-valued circuits. To do so, we use the following result to convert hardness against single-valued circuits into hitting sets for co-nondeterministic circuits.

Lemma 2.35 ([Uma03]). *There is a universal constant b and a deterministic polynomial-time algorithm that, on input 1^m and a truth table y of a function with single-valued circuit complexity at least m^b, outputs a set S of size $O(|y|^b)$ that hits co-nondeterministic circuits of size m that accept at least half of their inputs.*

We also need the following equivalence from [AvM17].

Lemma 2.36 ([AvM17]). $\Sigma_2\mathsf{EXP} \not\subseteq \mathsf{NP/poly}$ *if and only if* $\mathsf{prAM} \subseteq \text{io-}\Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$ *for all* $\epsilon > 0$.

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

Theorem 2.37. *Assume* $\Sigma_2\mathsf{EXP} \subseteq \mathsf{NP/poly}$. *Then* $\Sigma_2\mathsf{EXP}$ *has single-valued witnesses of polynomial size, i.e., for every* $L \in \Sigma_2\mathsf{EXP}$ *and linear-time (in its input length) co-nondeterministic verifier H for L, the following holds: For every $x \in L$, there exists a single-valued circuit C_x of size* $\mathsf{poly}(|x|)$ *such that $H(x, \cdot)$ accepts the exponential-length truth table of C_x.*

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

Assume that $\Sigma_2\mathsf{E}$ does not have single-valued witness circuits of size n^c for any constant c. This implies that for all $c \geq 1$, there is a co-nondeterministic verifier H_c that takes as input a string x and a string y of length $2^{O(|x|)}$, runs in time $2^{O(|x|)}$, and has the following property: For infinitely many n, there is a input x' of length n such that $H_c(x', y')$ accepts for some y', but *every* y accepted by $H_c(x', \cdot)$ has single-valued circuit complexity at least n^c. Thus, there are infinitely many n such that, if we give x' as n bits of advice, guess a string y of length $2^{O(n)}$, and verify that $H_c(x', y)$ accepts (using co-nondeterminism), we are guaranteed

that y encodes the truth table of a function with single-valued circuit complexity at least n^c. This gives us a Σ_2-procedure for obtaining hard functions, which we use to derandomize prAM and obtain a contradiction to Lemma 2.36.

Let $\Pi \in \mathsf{prAM}$ and let P be a protocol for Π that runs in time $O(\ell^a)$ on input length ℓ. By Proposition 2.17, to derandomize P it suffices to have a set S that hits any co-nondeterministic circuit of size $O(\ell^{2a})$ that accepts at least half of its inputs. To obtain such a set using Lemma 2.35, we need to first obtain a truth table of single-valued circuit complexity at least $\Omega(\ell^{2ab})$, where b is the constant from the lemma. Recall that our objective is to obtain a subexponential (time 2^{n^ϵ} for all $\epsilon > 0$) simulation. To this end, let $\epsilon > 0$ be sufficiently small and consider the verifier H_c for $c = 3ab/\epsilon$ on inputs of length $n = \ell^\epsilon$. If n is one of the infinitely many input lengths for which there exists x' such that every string accepted by $H_c(x', \cdot)$ has single-valued circuit complexity at least $n^c = \ell^{3ab}$, then we can obtain such a hard string by having x' as advice, guessing $y \in \{0,1\}^{2^{O(\ell^\epsilon)}}$ and verifying that $H_c(x', y)$ accepts.

In parallel, apply Lemma 2.35 to y to obtain a set S of size $2^{O(\ell^\epsilon)}$, and use S to derandomize the prAM computation (guessing a Merlin response for each string in S). Finally, accept if and only if both $H_c(x', y)$ and the prAM simulation accept. All of this can be carried out in $\Sigma_2\mathsf{TIME}[2^{O(\ell^\epsilon)}]/\ell^\epsilon$. Since ϵ is an arbitrarily small constant and the simulation works for infinitely many input lengths ℓ, we obtain a contradiction to Lemma 2.36. ∎

Theorem 2.37 allows us to establish the following complexity class collapse in case $\Sigma_2\mathsf{EXP} \subseteq \mathsf{NP/poly}$. The corollary represents the role our easy witness result plays in the proof of Theorem 2.9.

Corollary 2.38. *If* $\Sigma_2\mathsf{EXP} \subseteq \mathsf{NP/poly}$, *then* $\mathsf{P}^{\Sigma_2\mathsf{EXP}} = \mathsf{EXP}$.

Proof. Under the hypothesis from the statement, we show that $\Sigma_2\mathsf{EXP} = \mathsf{coNEXP}$, which suffices by combining Lemma 2.36 and Lemma 2.30. The hypothesis and Lemma 2.36 guarantee the negation of $\mathsf{prAM} \subseteq \text{io-}\Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$ for all ϵ, which in turn implies the negation

of $\mathsf{prAM} \subseteq \text{io-}\mathsf{NTIME}[2^{n^\epsilon}]/n^\epsilon$ for all ϵ, and thus the contrapositive of Lemma 2.30 implies $\mathsf{EXP} = \mathsf{NEXP}$ and therefore $\Sigma_2\mathsf{EXP} = \mathsf{coNEXP} = \mathsf{EXP}$. Finally, we have $\mathsf{P}^{\Sigma_2\mathsf{EXP}} = \mathsf{P}^{\mathsf{EXP}} = \mathsf{EXP}$.

To show that $\Sigma_2\mathsf{EXP} = \mathsf{coNEXP}$, by padding, it suffices to show that every $L \in \Sigma_2\mathsf{E}$ is in coNEXP. Fix $L \in \Sigma_2\mathsf{E}$. By Theorem 2.37, L has single-valued witnesses of size n^c for some constant c. On input $x \in \{0,1\}^n$, we cycle through all nondeterministic circuits C of size n^c and compute their truth tables in time $O(2^{n^c})$. For each truth table T, we then run $V(x,T)$ (where V is a co-nondeterministic verifier for L), accepting if and only if some verification accepts. All of this runs in exponential co-nondeterministic time, so we are done. ■

2.7.2 Simulation

We now execute our win-win strategy and establish Theorem 2.9 and its strengthening for $\mathsf{prBPP}_{||}^{\mathsf{SAT}}$ in lieu of prAM. We first consider the case where $\Sigma_2\mathsf{EXP} \not\subseteq \mathsf{NP/poly}$. In this case simulations of the required type that work on all inputs of a given length are provided by Lemma 2.36 for prAM. We argue the same simulations follow for $\mathsf{prBPP}_{||}^{\mathsf{SAT}}$.

Lemma 2.39. *If* $\Sigma_2\mathsf{EXP} \not\subseteq \mathsf{NP/poly}$*, then for every* $\epsilon > 0$

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

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

Our approach is to use Lemma 2.35 to derandomize the queries made to Γ while making sure that the overall simulation of M can be carried out in subexponential Σ_2-time. To derandomize Γ at input length $O(n^k)$ using the lemma, we need to obtain a truth table of single-valued circuit complexity at least $\Omega(n^{2bk})$, where b is the constant from the lemma. Let $\epsilon > 0$ and $L \in \Sigma_2\mathsf{E}$ be a language that has nondeterministic circuit complexity at least $n^{3bk/\epsilon}$ for infinitely many input lengths (which is guaranteed to exist by the hypothesis of

the theorem). The simulation of M on inputs x goes as follows: Given as advice the number of strings of length n^ϵ that are in L, the Σ_2 algorithm guesses the truth table of L at input length n^ϵ, verifies it, and uses it as the string y in Lemma 2.35. More precisely, after guessing the truth table, the algorithm performs the following operations in parallel:

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

- It guesses which of the queries to Γ that M makes on input x are answered positively and which are answered negatively. For each query that is guessed to be answered positively, it uses the set S from Lemma 2.35 and the existential phase to verify that there is a random-bit string in S for which Merlin can provide a witness. Similarly, it uses S and the universal phase to verify each query that is guessed to be answered negatively.

We note that the only existential computation paths that survive the computation are the ones where the truth table of L at input length n^ϵ was guessed correctly. In this case, and in the case that n^ϵ is one of the infinitely many input lengths where L has nondeterministic circuit complexity at least $n^{3bk/\epsilon}$, it holds that the guessed truth table has high enough (single-valued) nondeterministic circuit complexity such that S hits the co-nondeterministic circuits given by Proposition 2.17 for negative instances of Γ at input length $O(n^k)$. This further guarantees that the surviving existential computation paths are those that correctly guess the answers to all queries M makes on input x that are in the promise of Γ. This suffices to obtain a simulation of M that is correct on infinitely many input lengths since M is insensitive to variations in answers to queries that are outside the promise (even when the same query is answered differently on different occasions). Finally, we note that the entire

procedure runs in time $2^{O(n^\epsilon)}$, which can be made smaller than $2^{n^{\epsilon'}}$ for any $\epsilon' > 0$ by taking ϵ to be sufficiently small. ■

The other case of the win-win analysis is when $\Sigma_2\mathsf{EXP} \subseteq \mathsf{NP/poly}$. In this case, we use the collapse $\mathsf{P}^{\Sigma_2\mathsf{EXP}} = \mathsf{EXP}$ given by Corollary 2.38 and our targeted hitting-generator construction to obtain the desired simulation. We conclude:

Theorem 2.40 (Strengthening of Theorem 2.9). *For every $\epsilon > 0$ and every $e > 0$*

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

Proof. If $\Sigma_2\mathsf{EXP} \notin \mathsf{NP/poly}$, then it follows that $\mathsf{prBPP}_{\|}^{\mathsf{SAT}} \subseteq \Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$ for all $\epsilon > 0$ by Lemma 2.39. Otherwise, by Corollary 2.38, we have that $\mathsf{P}^{\Sigma_2\mathsf{EXP}} = \mathsf{EXP}$. By Theorem 2.32, all we need to show is that $\mathsf{P}^{\Sigma_2\mathsf{EXP}} \notin \bigcup_{b\in\mathbb{N}} \mathsf{BPTIME}[n^{b((\log n)^2)}]_{\|}^{\mathsf{SAT}}$. Given the containment $\mathsf{prBPP}_{\|}^{\mathsf{SAT}} \subseteq \mathsf{P}_{\|}^{\mathsf{prAM}}$ and a padding argument, it follows that $\bigcup_{b\in\mathbb{N}} \mathsf{BPTIME}[n^{b((\log n)^2)}]_{\|}^{\mathsf{SAT}} \subseteq \mathsf{DTIME}[2^{\mathsf{polylog}(n)}]_{\|}^{\mathsf{prAM}}$. It remains to show that $\mathsf{P}^{\Sigma_2\mathsf{EXP}} \notin \mathsf{DTIME}[2^{\mathsf{polylog}(n)}]_{\|}^{\mathsf{prAM}}$, which we do by diagonalization.

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

To conclude, we also state the following theorem, which extends the unconditional simulation for AM of [Wil16] to $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ and thus also to prAM.

Theorem 2.41. *There exists a constant c such that for every $\epsilon > 0$*

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

Proof. As before, if $\Sigma_2\mathsf{EXP} \not\subseteq \mathsf{NP/poly}$, then it follows that $\mathsf{prBPP}_{\|}^{\mathsf{SAT}} \subseteq \Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$ for all $\epsilon > 0$ by Lemma 2.39. Otherwise, if $\Sigma_2\mathsf{EXP} \subseteq \mathsf{NP/poly}$, then in particular $\Pi_2\mathsf{EXP} \subseteq \mathsf{coNP/poly}$. Because $\Pi_2\mathsf{E}$ has a complete problem under linear-time reductions, it also follows that there exists a constant k such that $\mathsf{prAM} \subseteq \Pi_2\mathsf{E} \subseteq \mathsf{coNTIME}[n^k]/n^k$.

We show that the last inclusion implies that there exists a constant c such that $\mathsf{prBPP}_{\|}^{\mathsf{SAT}} \subseteq \mathsf{P}_{\|}^{\mathsf{prAM}} \subseteq \mathsf{P}^{\mathsf{SAT}}/n^c \subseteq \cap_{\epsilon>0}\Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^c$. Let $\Pi \in \mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ and M be a polynomial-time probabilistic machine with non-adaptive oracle access to a prAM-complete problem Γ. The language $L_\Pi = \{(x, i) \mid \text{the } i\text{-th query made by } M \text{ on input } x \text{ is true}\}$ is in prAM, which we know is contained in $\mathsf{coNTIME}[m^k]/m^k$, where $m \leq 2n$ w.l.o.g. is the input length for L_Π when x has length n. To decide Γ with non-adaptive oracle access to SAT, it suffices to query the SAT oracle, for each i, on whether the polynomial-time co-nondeterministic simulation of L_Π accepts (x, i) with the correct advice string, and then run M on input x and the answers to the queries. This simulation can be carried out in polynomial time and with at most $(2n)^k \leq n^c$ bits of advice for a constant $c > k$ and sufficiently large n, so we are done. ∎

2.8 Further research

The most important problem that is left open is establishing a full equivalence between almost-all-inputs hardness and derandomization of prAM. Recall that to conclude $\mathsf{prAM} \subseteq \mathsf{NP}$, we require the existence of a length-preserving function $f \in \mathsf{NTIME}[n^a]$ such that f is hard on almost-all inputs against $\mathsf{prAMTIME}[n^{O((\log a)^2)}]$ (Theorem 2.5). However, our results in the other direction (Proposition 2.28) conclude the existence, for sufficiently large

a, of a length-preserving function $f \in \mathsf{NTIME}[n^a]$ with very few bits of advice such that f is hard on almost-all inputs against $\mathsf{AMTIME}[n^{O((\log a)^2)}]$. The remaining gaps are the advice and the technical difference between AM and prAM multi-bit protocols (see the discussion at the end of Section 2.5.1 for an explanation).

In the next chapter, we obtain full equivalences (with respect to leakage-resilient hardness) by relaxing the derandomization requirement to a *mild* derandomization, i.e., simulations in Σ_2-machines. In doing so, we also establish an equivalence between mild derandomization and the existence of targeted hitting-set generators for prAM that recover the corresponding mild derandomization. In Chapter 4, we improve the targeted generator construction in this chapter, and obtain a hardness assumption that fully characterizes derandomization for prAM via targeted generators.

Chapter 3

Mild Derandomization

3.1 Introduction

In the previous chapter, we mentioned briefly that Liu and Pass managed to obtain an equivalence between time-bounded derandomization ($\mathsf{prBPP} \subseteq \mathsf{P}$) and hardness in the presence of efficiently-computable leakage [LP23]. In this chapter, we explore this hardness notion and obtain equivalences with *mild* derandomization for prAM, i.e., simulation such as $\mathsf{prAM} \subseteq \Sigma_2\mathsf{P}$. We begin by defining the notion of leakage-resilient hardness for computational problems (multi-bit functions, or more generally relations $R \subseteq \{0,1\}^* \times \{0,1\}^*$). Intuitively, a leakage-resilient hard function is a function that remains hard to compute, for example on input x, even in the presence of a short, efficiently computable string that is output by an algorithm that gets to see the value of $f(x)$. We define a version of the notion in the setting of hardness on almost-all inputs.

Definition 3.1 (Leakage-resilient hardness). A relation R is (T, ℓ)-leakage-resilient hard on almost-all inputs against a class of algorithms if for all pairs (Leak, A) of algorithms in the class that run in time T and such that $|\mathrm{Leak}(x, y)| \leq \ell(|x|)$, the following holds for almost-all inputs x and every $y \in R(x) = \{y \mid (x, y) \in R\}$: The probability that $A(x, \mathrm{Leak}(x, y)) = y$ is at most $1/3$. ◂

In cryptographic terms, the algorithm A can be viewed as an attacker that attempts to compute $y \in R(x)$ by receiving a few bits of information about y from the leakage-providing algorithm Leak. In some cases, such as when the relation R is computable in P, one may take R to be a function.

At the high end of the derandomization spectrum, the main result of Liu and Pass [LP23] establishes an equivalence between the derandomization $\mathsf{prBPP} \subseteq \mathsf{P}$ and the existence of a length-preserving function $f \in \mathsf{P}$ that is leakage-resilient hard on almost all inputs against probabilistic algorithms running in fixed-polynomial time and with sublinear leakage. The equivalence works across the entire derandomization spectrum. For completeness we mention that, in another work, Liu and Pass establish an equivalence between derandomization and hardness related to approximating conditional Levin-Kolmogorov complexity (Kt) [LP22]. The latter equivalence does not scale well toward the low end, though.

One can also ask about efficient simulations of BPP on *nondeterministic* machines. Similar to the AM setting, we refer to such simulations as *mild* derandomizations. Establishing mild derandomization results for prBPP is a required step if we hope to show stronger deterministic simulations for the class. At the low-end of the derandomization spectrum, an equivalence is known between whitebox and blackbox mild derandomization for prBPP that work for infinitely-many input lengths [IKW02]. Specifically, a subexponential-time *nondeterministic* simulation for prBPP with subpolynomial advice that works for infinitely-many input lengths is equivalent to polynomial-size circuit lower bounds for the class NE. Formally, $\mathsf{prBPP} \subseteq \text{io-}\mathsf{NTIME}[2^{n^\epsilon}]/n^\epsilon$ for all $\epsilon > 0$ if and only if $\mathsf{NE} \not\subseteq \mathsf{NP/poly}$ [IKW02]. The technique, however, does not scale and is only known to work in the infinitely-often setting. The Liu-Pass result on the connection between leakage-resilient hardness and derandomization extends to the mild setting. In contrast to the low-end equivalence of [IKW02], Theorem 3.2 extends to the entire derandomization spectrum. We state the high-end version.

Theorem 3.2 (Follows from [LP23, Theorem 1.6]). *There exists a constant c such that for all $\epsilon \in (0,1)$, the following are equivalent.*

- $\mathsf{prBPP} \subseteq \mathsf{NP}$.
- *There exists a total length-preserving relation* $R \in \mathsf{NP}$ *that is* (n^c, n^ϵ)*-leakage-resilient hard on almost-all inputs against* prBPP.

AM setting. While it is believed that $\mathsf{prAM} \subseteq \mathsf{NP}$ and we know that $\mathsf{prAM} \subseteq \Pi_2\mathsf{P} = \mathsf{coNP}^{\mathsf{NP}}$, it remains open whether $\mathsf{prAM} \subseteq \mathsf{P}^{\mathsf{NP}}$ or even whether $\mathsf{prAM} \subseteq \Sigma_2\mathsf{P} = \mathsf{NP}^{\mathsf{NP}}$, both of which are required steps toward showing that $\mathsf{prAM} \subseteq \mathsf{NP}$. We note that even a subexponential-time Σ_2-simulation with subpolynomial advice for prAM that works for infinitely many input lengths remains open. In this setting, an equivalence between whitebox and blackbox derandomization is known: The simulation is equivalent to polynomial size *nondeterministic* circuit lower bounds for the class $\Sigma_2\mathsf{E}$. In symbols, $\mathsf{prAM} \subseteq \text{io-}\Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$ for all ϵ if and only if $\Sigma_2\mathsf{E} \notin \mathsf{NP/poly}$ [AvM17]. The equivalence does not scale well, and it is unknown whether it holds for other parameter settings such as the high end.

Our results. Our main contribution for this chapter is establishing *full* equivalences for *mild* derandomization of prAM. Specifically, we obtain an equivalence between mild derandomization of prAM and leakage-resilient hardness on almost-all inputs against $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$, the class of polynomial-time probabilistic algorithms with non-adaptive oracle access to SAT. The equivalence scales smoothly across the entire derandomization spectrum, and applies to other classes between NP and $\Sigma_2\mathsf{P}$ in addition to $\Sigma_2\mathsf{P}$. We state the high-end version.

Theorem 3.3. *Let* $\mathcal{C} \in \{\mathsf{P}^{\mathsf{NP}}, \mathsf{ZPP}^{\mathsf{NP}}, \Sigma_2\mathsf{P}\}$. *There exists a constant* c *such that for all* $\epsilon \in (0,1)$*, the following are equivalent.*

- $\mathsf{prAM} \subseteq \mathcal{C}$.
- *There exists a total length-preserving relation* $R \in \mathcal{C}$ *that is* (n^c, n^ϵ)*-leakage-resilient hard on almost-all inputs against* $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$.

Theorem 3.3 can be viewed as a counterpart to Theorem 3.2 in the mild setting for prAM. Indeed, when compared to the traditional setting, both require an extra level of nondeterminism in the hardness assumption as well as in the simulation.

As seen in Chapter 2, the class $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ was used in the initial derandomization results for Arthur-Merlin protocols, and derandomization results for prAM typically translate into corresponding derandomization results for $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$. In particular, the derandomization $\mathsf{prAM} \subseteq \Sigma_2\mathsf{P}$ of Theorem 3.3 implies that $\mathsf{prBPP}_{\|}^{\mathsf{SAT}} \subseteq \Sigma_2\mathsf{P}$. Also, lower bounds for linear-exponential time classes such as E and NE against nondeterministic circuits (which suffice for derandomizing prAM) imply non-uniform lower bounds for the same class against deterministic circuits with non-adaptive oracle access to SAT (which suffice for derandomizing $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$) [SU06].

We show that a connection between $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ and prAM holds in the leakage-resilient hardness setting as well. Specifically, we show that the derandomization conclusion of Theorem 3.3 holds under a weaker hardness assumption on *learn-and-evaluate* Arthur-Merlin protocols. As seen in Section 2.3.3 learn-and-evaluate protocols are a two-phase type of Arthur-Merlin protocol that arises naturally in hardness vs. randomness tradeoffs and fits nicely into the leakage-resilient hardness paradigm. In the first phase, a probabilistic algorithm with oracle access to a function f produces a short sketch π that consists of evaluations of f. In our case, f is the function that maps an index i to the i-th bit of a string y such that $(x, y) \in R$, and the first phase can be viewed as a small amount of leakage on y. In the second phase, an Arthur-Merlin protocol computes f given π, which naturally translates into a protocol that attempts to recover y from a "small" amount of leakage. The result, taken together with the derandomization-to-hardness direction of Theorem 3.3, implies an equivalence between leakage-resilient hardness on almost-all inputs against $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ and learn-and-evaluate protocols.

Theorem 3.4. *Let $\mathcal{C} \in \{\mathsf{P}^{\mathsf{NP}}, \mathsf{ZPP}^{\mathsf{NP}}, \Sigma_2\mathsf{P}\}$. There exists a constant c such that for all $\epsilon \in (0, 1)$, the following are equivalent.*

1. *There exists a total length-preserving relation $R \in \mathcal{C}$ that is (n^c, n^ϵ)-leakage-resilient hard on almost-all inputs against* $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$.

2. *There exists a total length-preserving relation $R \in \mathcal{C}$ that is (n^c, n^ϵ)-leakage-resilient hard on almost-all inputs against learn-and-evaluate protocols.*

Theorem 3.4 partially addresses an open problem posed by Shaltiel and Umans of obtaining uniform equivalences between hardness against Arthur-Merlin protocols and algorithms with non-adaptive oracle access to SAT [SU06]. They ask whether an implication such as $\mathsf{E} \notin \mathsf{AM} \implies \mathsf{E} \notin \mathsf{P}_{\|}^{\mathsf{SAT}}$ holds. Theorem 3.4 establishes an analogous result in the mild leakage-resilient hardness setting.

We also address, in the mild setting, the open problem posed by Goldreich about AM [Gol11]. We show that mild derandomization of prAM is equivalent to the existence of targeted hitting-set generators achieving the same derandomization. We state such a result at the high end of the derandomization spectrum, but it extends to the entire range.

Theorem 3.5. *Let $\mathcal{C} \in \{\mathsf{P}^{\mathsf{NP}}, \mathsf{ZPP}^{\mathsf{NP}}, \Sigma_2\mathsf{P}\}$. If $\mathsf{prAM} \subseteq \mathcal{C}$, then there exist targeted hitting-set generators that achieve this mild derandomization result.*

In fact, we show that the assumption of Theorem 3.5 implies the existence of targeted *pseudorandom* generators for non-adaptive SAT-oracle circuits. Employing Theorem 3.5, we also obtain a simple proof for the result that if $\mathsf{prAM} \subseteq \mathsf{P}^{\mathsf{NP}}$, then E^{NP} has maximum circuit complexity [AGH+11] (see the end of Section 3.6).

Finally, we connect leakage-resilient hardness to the known equivalence between whitebox and blackbox mild derandomization of prAM at the low end. As mentioned before, the simulation $\mathsf{prAM} \subseteq \text{io-}\Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$ for all $\epsilon > 0$ is equivalent to the non-uniform lower bound $\Sigma_2\mathsf{E} \notin \mathsf{NP/poly}$ [AvM17]. We show that those conditions are also equivalent to leakage-resilient hardness where the algorithm A needs to produce each individual bit of a solution $y \in R(x)$ very efficiently. The equivalence of [AvM17] holds in the infinitely-often setting.

Correspondingly, the equivalent leakage-resilient hardness condition holds for all inputs of infinitely-many input lengths.

Definition 3.6 (Local Leakage-resilient hardness). A relation R is t-local (T, ℓ)-leakage-resilient hard on all inputs of infinitely-many input lengths against a class of algorithms if for all pairs (Leak, A) of algorithms in the class such that Leak runs in time T, $|\text{Leak}(x, f(x))| \leq \ell(|x|)$ and A runs in time t, the holds for infinitely many $n \in \mathbb{N}$ every $x \in \{0,1\}^n$ and every $y \in R(x)$ and every $i \in [|y|]$: The probability that $A(x, \text{Leak}(x, y), i) = y_i$ is at most $1/3$. ◂

This definition allows for separate running times for the leakage-providing algorithm Leak and the algorithm A, and in particular allows A to run in time that is sublinear in the length of a solution $y \in R(x)$.

Theorem 3.7. *The following are equivalent:*

1. $\mathsf{prAM} \subseteq \text{io-}\Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$ *for all* $\epsilon > 0$.
2. $\Sigma_2\mathsf{E} \notin \mathsf{NP/poly}$.
3. *For all* $\epsilon > 0$ *there exists a total relation* $R \in \Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$ *that is* $\mathsf{poly}(n)$*-local* $(2^{n^\epsilon}, \mathsf{poly}(n))$*-leakage-resilient hard on all inputs of infinitely-many input lengths against* $\mathsf{prBPP}_{||}^{\mathsf{SAT}}$.
4. *For all* $\epsilon > 0$ *and* $c \geq 1$ *there exists a total length-preserving relation* $R \in \Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$ *that is* n^c*-local* $(n^c, \ell(n))$*-leakage resilient hard on all inputs of infinitely-many input lengths against* $\mathsf{prBPP}_{||}^{\mathsf{SAT}}$*, where* $n^{\Theta(1)} \leq \ell(n) \leq n - \omega(1)$ *is polynomial-time computable.*

We only know Theorem 3.7 for the low-end of the derandomization spectrum due to the use of [AvM17].

Techniques. We combine the approach of Liu and Pass in the leakage-resilient setting for BPP (see Section 3.2 for an overview) with constructions that are suitable for the AM setting.

In the hardness-to-derandomization direction, given a hard relation R, we use the value of $y \in R(x)$ as a basis for instantiating the nondeterministic version of the Shaltiel-Umans (SU) generator for prAM [SU05]. As the SU generator is originally meant for the the non-uniform setting, the original reconstruction argument only guarantees the existence of a small nondeterministic circuit that computes y in case the generator fails. Thus, in order to use the construction we need to "uniformize" the reconstruction procedure and show that such a circuit can be constructed by an efficient algorithm with non-adaptive oracle access to SAT. To obtain the results under hardness against learn-and-evaluate protocols, we employ the same approach but instead use the recursive version of the Miltersen-Vinodchandran generator [MV05] that we dubbed RMV in Chapter 2 and defined in Section 2.4.1. RMV does not scale as well as SU but still suffices for obtaining Theorem 3.4.

For the derandomization-to-hardness direction, similar to [LP23], we frame the problem of computing a leakage-resilient hard function as a $\mathsf{BPP}_{\|}^{\mathsf{SAT}}$ search problem, and then make use of a search-to-decision reduction à la [Gol11] together with the derandomization assumption. We first show that for a fixed x, a random choice r of $y \in R(x)$ suffices: With high probability, the first n algorithms Leak and A according to some canonical enumeration fail to compute r in the sense that $A(x, \mathrm{Leak}(x, r)) \neq r$. Then we show that checking whether a candidate r for $R(x)$ is hard (again w.r.t. the first n algorithms) can be done by a $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ algorithm. To conclude the argument for $R \in \Sigma_2\mathsf{P}$, we guess-and-verify a "good" value of $y \in R(x)$ by exploiting the connection between $\mathsf{BPP}_{\|}^{\mathsf{SAT}}$ and prAM [CR11] together with the derandomization assumption on prAM.

We remark that both directions of the Liu-Pass result for prBPP relativize, which immediately implies an equivalence between the derandomization $\mathsf{prBPP}^{\mathsf{SAT}} \subseteq \mathsf{P}^{\mathsf{SAT}}$ and the existence of leakage-resilient hard functions computable in $\mathsf{P}^{\mathsf{SAT}}$ that are hard on almost-all inputs against probabilistic algorithms with SAT oracle. Since our objective is to obtain equivalences with respect to derandomizing prAM, and it is unknown whether (mild) derandomization of prAM implies derandomization of $\mathsf{prBPP}^{\mathsf{SAT}}$, the relativization approach is

insufficient for our purposes.

Organization. In Section 3.2, we develop the ideas behind our results and relate them to existing techniques. We start the formal treatment in Section 3.3 with definitions, notation, and other preliminaries. We present the uniformization for the SU reconstructor in Section 3.4. In Section 3.5, we develop the equivalence between leakage-resilient hardness on almost-all inputs and mild derandomization of prAM (Theorem 3.3). In Section 3.5.4, we develop the equivalence between leakage-resilient hardness on almost-all inputs against $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ and against learn-and-evaluate protocols (Theorem 3.4). Section 3.5.3 contains the equivalence between mild derandomization of prAM and the existence of targeted hitting-set generators (Theorem 3.5). In Section 3.6, we discuss local leakage resilience (Theorem 3.7).

3.2 Technical overview

In this section, we present an overview of the techniques underlying our results. In order to keep the description self-contained, we start with a brief overview of the learning reconstructive paradigm used in prior hardness vs. randomness tradeoffs for BPP and AM. We then provide an overview of the hardness-to-derandomization direction of the Liu-Pass result in a way that facilitates the transition to our results on the mild derandomization setting for AM.

Learning reconstructive paradigm. pseudorandom generator constructions G typically have a hard function h as a basis (where how "hard" h is depends on the context) and produce a pseudorandom distribution G^h that depends on h. The proof of correctness for such generators is *reconstructive*: It presents an algorithm that, given access to any process D that distinguishes the distribution G^h from a truly random distribution, as well as to some additional information a, computes the hard function h. It is common for the additional information a to be composed of evaluations of h at a small number of points, in which

case we also say that the reconstruction is *learning*. Thus, unless G^h "fools" an efficient randomized process P on input x, the function h can be reconstructed efficiently from the distinguisher $D(r) \doteq P(x, r)$ and the evaluations.

Targeted setting. In Chapter 2, we saw that one way to obtain a targeted PRG is to use a hardness-based (oblivious) pseudorandom generator construction and instantiate it with a function h_x that depends on the input x. In the leakage-resilient setting and given a hard function f, we take h_x to be the function whose truth-table is $f(x)$. The leakage-providing algorithm Leak uses access to $f(x)$ to provide the answers to the learning queries, which are then fed into the algorithm A to compute h_x and thus $f(x)$.

Liu and Pass [LP23] employ the Nisan-Wigderson (NW) pseudorandom generator construction [NW94] combined with the locally-list-decodable encoding of [STV01], and instantiate the PRG with the function $h_x : \{0,1\}^{\log n} \to \{0,1\}$ that computes the mapping $i \mapsto f(x)_i$. In the reconstruction, the leakage-providing algorithm Leak uses access to h_x to answer the learning queries for the NW reconstruction algorithm as well to identify a "good" random string to be used in the list-decoding step and the position of h_x in the resulting list. This allows the algorithm A to compute the value of $f(x)$ by running the NW reconstruction followed by the list-decoding process with the "good" random string, outputting the element in the correct position of the resulting list.

In porting the Liu-Pass approach to the mild derandomization setting for prAM, we follow a similar idea but replace the NW construction by other hardness-based pseudorandom generator (PRG) or hitting-set generator (HSG) constructions that exhibit the learning property: In the case of failure of the PRG/HSG on an input x, the underlying hard function h can be reconstructed efficiently from the values of h at a small number of points. In particular, we make use of the following HSGs that have the learning property and are tailored for Arthur-Merlin protocols:

- The construction by Shaltiel and Umans [SU09], which we dub SU and use to derive

our main results under leakage-resilient hardness against $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$. This construction scales throughout the entire derandomization spectrum and, in particular, allows for obtaining tighter connections with blackbox derandomization via our local-leakage-resilient hardness results. To employ this generator, we present a uniform version for its reconstruction.

- The recursive version of the construction by Miltersen and Vinodchandran [MV05] due to Shaltiel and Umans [SU09], as seen in Section 2.4.1, which we use to derive our results under hardness against learn-and-evaluate protocols.

The SU construction scales better than RMV toward the low end, and thus we choose to employ SU for our main results. The disadvantage is that SU requires hardness against $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ instead of against Arthur-Merlin protocols. However, as evidenced in Theorem 3.3, such a hardness assumption is also required in the mild setting.

We now describe the SU construction in more detail and how we employ it to obtain our results. In the interest of keeping this section self-contained, we also review the RMV construction.

Shaltiel-Umans construction. The SU generator takes any low-degree polynomial $\hat{g} : \mathbb{F}^r \to \mathbb{F}$, which is usually obtained via a low-degree extension/Reed-Müller encoding (in our case, of $y \in R(x)$ of length n), and produces a set of strings of length m, where m is a parameter. The reconstruction for the generator shows the existence of a small (of size $\mathsf{poly}(m, \log n)$) nondeterministic circuit computing $\hat{g}$ in case the construction fails as a hitting-set generator.

The original reconstruction for the generator requires access to a few pieces of information in relation to the field $\mathbb{F}^r$ and the polynomial $\hat{g}$: A generating matrix M for the set $\mathbb{F}^r \setminus \{\vec{0}\}$, two "good" low-degree curves C_1 and C_2 over $\mathbb{F}^r$ and evaluations of $\hat{g}$ over points that depend on M, C_1 and C_2. For our purposes, the leakage-providing algorithm can compute the matrix M efficiently enough via a brute-force search. As for the curves C_1 and C_2, Shaltiel and

Umans show that a random choice has the required properties with high probability, which allows Leak to sample those at random. Finally, the evaluations of $\hat{g}$ can be computed efficiently given access to the value of $y \in R(x)$ by having Leak compute its low-degree extension $\hat{g}$.

In the nondeterministic setting, there is an extra complication: The reconstructor also requires oracle access to a list of r predictors $P_1, \ldots, P_r$, all of which can be obtained from a distinguisher D for the generator and are run over random curves similar to C_1 and C_2. As the predictors themselves are nondeterministic, there is the issue of complementation: How can negative predictions be computed nondeterministically, instead of with an oracle for an NP-complete problem such as SAT? Shaltiel and Umans use a strategy (originating in [FF93]) to approximately and nondeterministically evaluate the predictors. The strategy requires for each predictor an approximation $\tilde{p}_i$ up to $O(\log n)$ bits of precision of the probability p_i that the prediction is positive over a random choice of inputs for the generator. The high-level idea is as follows: Say we know the approximate probability $\tilde{p}_i$ that the nondeterministic predictor P_i is positive, and want to compute k predictions over a random curve. With high probability, close to $\tilde{p}_i \cdot k$ of the predictions are positive, and thus we can guess witnesses for a little less than $\tilde{p}_i \cdot k$ positive predictions, and assume that the remaining predictions are negative. This may mean that a small fraction of the predictions are incorrect, but because of the use of error correction, the reconstruction argument is robust against such a small amount of error.

Shaltiel and Umans provide the approximations $\tilde{p}_i$ as non-uniform additional input to the reconstructor. Recall that our objective is to obtain a uniform version of the reconstructor, but we can afford non-adaptive oracle access to SAT. We cannot simply use the SAT oracle to evaluate the predictors because the inputs for later predictions depend on the results of previous predictions, which would result in adaptive access to the SAT oracle. Instead, we exploit the fact that the SU generator is efficiently computable given access to the basis polynomial $\hat{z}$, and use the SAT oracle to efficiently obtain the approximations required to

run the original reconstructor. This only requires nonadaptive access to the SAT oracle because it only involves evaluating the predictors on uniformly-random outputs for the SU generator, and each such evaluation can be performed in parallel. In the end, we obtain a version A_{rec} of the reconstruction that, on input the values of the learning queries output by Leak (as well as additional information such as the curves C_1, C_2 and the matrix M), which have length $\mathsf{poly}(m, \log n)$, runs in time $\mathsf{poly}(m, \log n)$, computes $\hat{z}$ with high probability and only makes non-adaptive SAT oracle queries. By picking a small value of m, the amount of leakage is indeed small with relation to $|y| = n$.

Recursive Miltersen-Vinodchandran construction. The RMV generator also takes any low-degree polynomial $\hat{g} : \mathbb{F}^r \to \mathbb{F}$ and a parameter m and produces a set of strings of length m.

The RMV reconstructor for $\hat{g}$ forms a *commit-and-evaluate protocol*, a notion introduced in [SU09] for this reason. It is an Arthur-Merlin protocol that consist of two phases. The first phase is the commitment phase, where Arthur and Merlin interact to produce a commitment π. The commitment is then given as input to the evaluation phase, in which Arthur and Merlin compute evaluations of a function g_π that is determined by π and supposedly equals $\hat{g}$. As Merlin could cheat in the commitment phase, a key property of the protocol is its *resilience*: Given a commitment π, the evaluation protocol can only produce evaluations of a fixed function g_π, except with low probability. As g_π might differ from $\hat{g}$, one usually combines the construction with a checker (as in [GST03]) or a PCP (as we did in Chapter 2) to verify that $g_\pi = \hat{g}$.

In the commitment phase, Arthur samples a small (of size $\mathsf{poly}(m, \log n)$) set of points in $\mathbb{F}^r$ and the honest Merlin provides evaluations of $\hat{g}$ at those points as the commitment π. With high probability over the set chosen by Arthur, given the correct commitment π as additional input and access to a distinguisher D, the evaluation protocol computes $\hat{g}$ with high confidence, no matter what strategy Merlin uses.

By replacing the interaction in the commitment phase by a probabilistic algorithm with oracle access to $\hat{g}$, the RMV reconstruction can be cast as a *learn-and-evaluate protocol*. A learn-and-evaluate protocol is a reconstructor for a function h that consists of two phases, where only the second phase requires access to the distinguisher D. The two phases are:

1. A learning phase, which is a randomized algorithm that can make queries to h and outputs a string, which we refer to as a sketch.

2. An evaluation phase, which is a promise Arthur-Merlin protocol with access to D that takes a sketch and an evaluation point z as input, and is supposed to output $h(z)$.

With high probability, given the correct answers to the queries, the learning phase should output a sketch such that the evaluator with access to a distinguisher D is complete and sound.

The learn-and-evaluate version of the reconstructor naturally fits the leakage-resilient hardness-to-derandomization framework, where the leakage-providing algorithm Leak is a probabilistic algorithm with access to $y \in R(x)$ that samples the set of points and produces evaluations of the low-degree extension $\hat{g}$ of y on those points and the algorithm A is an Arthur-Merlin protocol that takes those evaluations as additional input to compute y. The amount of leakage is upper bounded by $\mathsf{poly}(m, \log n)$, which just like in the SU construction can be made small in relation to $|y| = n$ by picking a sufficiently small m.

3.3 Preliminaries

In this section, we present preliminary definitions and results that are necessary for developing our contributions. We start by extending the definition of targeted hitting-set generators for prAM (Definition 2.16) to accommodate computational models such as Σ_2-machines, then present some results on the class $\mathsf{BPP}_{\|}^{\mathsf{SAT}}$. The results in the chapter also rely on the preliminaries for the previous chapter.

3.3.1 Targeted hitting-set generators

To accommodate different computational models for *generating* the hitting sets (including nondeterministic ones), in this chapter we define targeted generators based on a relation between a circuit D and a hitting set S. In the following definition, we let $\mathcal{C}$ denote a machine model and $\mathcal{C}\mathsf{TIME}[T]$ denote computational problems computable by machines in $\mathcal{C}$ that run in time T.

Definition 3.8. Let H be an algorithm that computes a relation $R \in \mathcal{C}\mathsf{TIME}[T(m)]$ between co-nondeterministic circuits of size m and sets of strings of length m. We say that H is a targeted hitting-set generator for prAM computable in $\mathcal{C}\mathsf{TIME}[T(m)]$ if the following two conditions hold for all sufficiently large $m \in \mathbb{N}$:

- For all co-nondeterministic circuits D of size m, there exists a non-empty S such that $(D, S) \in R$.
- For all co-nondeterministic circuits D of size m that accept at least a $1/2$ fraction of their inputs, it holds that for every $S \in R(D)$ there exists $\rho \in S$ such that $D(\rho)$ accepts.

◂

Notice that we have the targeted hitting-set generator take only the circuit D as input, which is sufficient for derandomization as discussed in Section 2.3.4.

For some classes of algorithms, such as nondeterministic algorithms, the notion of computing a relation is intuitive: Upon receiving an input, the algorithm eventually halts and produces an output in the relation on every accepting computation path. For other classes of algorithms, such as $\Sigma_2\mathsf{P}$, the notion of computing a relation is perhaps less intuitive. We say that a Σ_2-algorithm N computes a relation R if for all $x \in \{0,1\}^*$, any accepting configuration of $N(x)$ outputs a value y such that $(x, y) \in R$.

It might not be immediately clear that, for example, a targeted hitting-set generator for prAM computable in $\Sigma_2\mathsf{P}$ implies that $\mathsf{prAM} \subseteq \Sigma_2\mathsf{P}$. For future reference. we include a generic statement and proof in the Σ_2 setting.

Proposition 3.9. *Assume that there exists a targeted hitting-set generator H for co-nondeterministic circuits computable in $\Sigma_2\mathsf{TIME}[T(m)]$. Then $\mathsf{prAM} \subseteq \bigcup_{k\in\mathbb{N}} \Sigma_2\mathsf{TIME}[T(n^k)]$.*

Proof. Let $\Pi \in \mathsf{prAM}$, and let P be an Arthur-Merlin protocol (with perfect completeness) that runs in time n^k for some constant k and decides Π. It is standard that we can obtain from P and an input $x \in \{0,1\}^*$, in time $O(|x|^{2k})$, a co-nondeterministic circuit $D_{P,x}$ of size at most $|x|^{2k}$ such that:

$$x \in \Pi_Y \implies \Pr_r[D_{P,x}(r) = 1] = 0.$$
$$x \in \Pi_N \implies \Pr_r[D_{P,x}(r) = 1] \geq 1/2.$$

The Σ_2-simulation for Π works as follows on input x: First, it computes $D_{P,x}$, and guesses a set S output by $H(D_{P,x})$. Using another nondeterministic guess, the simulation verifies that $D_{P,x}$ rejects every $\rho \in S$, rejecting otherwise. In parallel, the simulation verifies, using an existential and a universal guess, that the guessed set S is an output of H for $D_{P,x}$, rejecting otherwise.

If $x \in \Pi_Y$, then $D_{P,x}$ rejects every $\rho \in S$ for any S output by $H(D_{P,x})$, in which case the overall simulation accepts. If $x \in \Pi_N$, then correctness of H implies that $D_{P,x}$ accepts some $\rho \in S$ for every S output by $H(D_{P,x})$, in which case the overall simulation rejects. The running time for the simulation is $T(n^{2k})$, which completes the proof. ∎

3.3.2 Non-adaptive oracle access to SAT

For completeness and due to the important role the class plays in this chapter, we define the class $\mathsf{prBPP}_{||}^{\mathsf{SAT}}$. We say that a promise problem $\Pi = (\Pi_Y, \Pi_N)$ is in $\mathsf{prBPP}_{||}^{\mathsf{SAT}}$ if there exists a probabilistic algorithm M with non-adaptive oracle access to SAT such that for all

$x \in \{0,1\}^*$:

$$x \in \Pi_Y \implies \Pr[M(x) = 1] \geq 2/3.$$
$$x \in \Pi_N \implies \Pr[M(x) = 1] \leq 1/3.$$

By non-adaptive oracle access, we mean that the queries made by M cannot depend on the answers to previous queries, i.e., the queries must all be made in parallel.

Recall that by Lemma 2.11, $\mathsf{prBPP}_{\|}^{\mathsf{SAT}} \subseteq \mathsf{P}_{\|}^{\mathsf{prAM}}$, a containment that we repeatedly use in this chapter.

3.4 Uniform version of the SU reconstructor

In this section, we present a uniform version of the original reconstruction algorithm for the nondeterministic SU generator due to Shaltiel and Umans [SU05]. Our objective is to establish the following result.

Lemma 3.10. *There exists a deterministic algorithm H_{det} and a pair $A_{rec} = (A_{comp}, A_{dec})$ consisting of a probabilistic algorithm A_{comp} and a deterministic algorithm A_{dec} with non-adaptive access to a SAT-oracle such that for every $z \in \{0,1\}^*$, $m \in \mathbb{N}$ and co-nondeterministic circuit D of size m, at least one of the following holds:*

1. *$H_{det}(z, 1^m)$ outputs a hitting set for D.*
2. *With probability at least $2/3$, $A_{dec}(A_{comp}(z, 1^m), D)$ outputs z.*

The construction also has the following properties:

- Compression: *On input z of length n and 1^m, the string output by A_{comp} has length $\mathsf{poly}(m, \log n)$.*
- Efficiency: *On input z of length n and D of size m, H runs in time $\mathsf{poly}(m, n)$. On input z of length n and 1^m, A_{comp} runs in time $n \cdot \mathsf{poly}(m, \log n)$, and A_{dec}, given the*

output of $A_{comp}(z, 1^m)$, D of size m and an additional index i, computes the i-th bit of z in time $\mathsf{poly}(m, \log n)$.

- Input access: *The only way A_{dec} requires access to D is via blackbox access to the deterministic predicate that underlies D.*

To prove Lemma 3.10, we begin by describing the SU construction. The basis for the nondeterministic Shaltiel-Umans hitting-set generator construction is a polynomial $\hat{g} : \mathbb{F}^r \to \mathbb{F}$ of total degree at most Δ, where $\mathbb{F} = \mathbb{F}_p$ for a prime p. To compute $\mathrm{SU}(\hat{g})$, we need access to a matrix M that generates $\mathbb{F}^r \setminus \{\vec{0}\}$, in the sense that $\{M^i \vec{v}\}_{1 \le i < p^r} = \mathbb{F}^r \setminus \{\vec{0}\}$ for any nonzero $\vec{v} \in \mathbb{F}^r$. A systematic way of doing this is as follows: First, we compute an irreducible polynomial $P(x) \in \mathbb{F}[x]$ of degree r and identify $\mathbb{F}^r$ with $\mathbb{F}[x]/(P(x))$. Then, we find a generator $G(x)$ for $\mathbb{F}[x]/(P(x)) \setminus \{0\}$ and set M as the linear transformation corresponding to multiplication by $G(x)$. Given oracle access to $\hat{g}$, finding $P(x)$ can be done in time $\mathsf{poly}(p, r)$ by using Shoup's algorithm [Sho88]. Once we have $P(x)$, [Sho92] gives us a $\mathsf{poly}(p, r)$ procedure that outputs a list of elements in $\mathbb{F}[x]/(P(x))$ that contains a generator. For our purposes, it then suffices to single-out a generator in the list by factoring $p^r - 1$ and computing the relevant powers of each element. Using a trivial factorization algorithm, this can be done in time $\sqrt{p^r} \cdot \mathsf{poly}(p, r)$. A similar strategy for computing M was used in [CLO+23] to obtain a uniform version for the regular (deterministic) Shaltiel-Umans generator.

Once M is fixed, define r functions $G^{(s)} : \mathbb{F}^r \to \mathbb{F}^m$ (where m is the length of strings we want our generator to output) for $0 \le s \le r - 1$, such that $G^{(s)}(\vec{y}) = \hat{g}(M^{p^s} \vec{y}) \circ \hat{g}(M^{2 \cdot p^s} \vec{y}) \circ \cdots \circ \hat{g}(M^{m \cdot p^s} \vec{y})$. We refer to each $0 \le s \le r - 1$ as a "stride" for the generator. We then transform each function into a binary one by encoding it with a suitable error-correcting code. We say that a code $E : \{0,1\}^k \to \{0,1\}^{n'}$ is (ρ, ℓ)-list-decodable if for all $x \in \{0,1\}^k$, the set $\{x \mid \text{the hamming distance of } E(x) \text{ and } r \text{ is at most } (1/2 - \rho)n'\}$ has size at most ℓ. For our purposes, it suffices that such a code exists for the parameters setting we need.

We take a (γ, γ^{-2})-list-decodable code $E : \{0,1\}^{\log p} \to \{0,1\}^{n'}$ (for a value $\gamma = \frac{1}{24m}$) with encoding length $n' = \mathsf{poly}(\log p, \gamma^{-1}) = \mathsf{poly}(\log p, m)$ and computable in time $\mathsf{poly}(n')$. Define $G'^{(s)} : \{0,1\}^{r \log p + \log n'} \to \{0,1\}^m$, where

$$G'^{(s)}(\vec{y}, j) = E(\hat{g}(M^{p^s}\vec{y}))_j \circ E(\hat{g}(M^{2 \cdot p^s}\vec{y}))_j \circ \cdots \circ E(\hat{g}(M^{m \cdot p^s}\vec{y}))_j.$$

Finally, the (seeded) hitting-set generator is the union of $G'^{(s)}$ for all s. Specifically, we have

$$H(\vec{y}, j, s) = E(\hat{g}(M^{p^s}\vec{y}))_j \circ E(\hat{g}(M^{2 \cdot p^s}\vec{y}))_j \circ \cdots \circ E(\hat{g}(M^{m \cdot p^s}\vec{y}))_j.$$

The SU (hitting-set) generator $\mathrm{SU}(\hat{g})$ is the function that outputs the union of $H(\vec{y}, j, s)$ over all $\vec{y} \in \mathbb{F}^r$, $j \in [n']$ and $0 \le s \le r-1$. Notice that, with access to the table of $\hat{g}$ and M, the construction runs in time $\mathsf{poly}(p^r, m)$ and outputs at most that many strings.

For the reconstruction, our starting point is a version of the standard uniform distinguisher-to-predictor reduction for co-nondeterministic circuits that is particularly suitable for distinguishers for the SU generator.

Lemma 3.11. *Fix a polynomial $\hat{g} : \mathbb{F}^r \to \mathbb{F}$ for $\mathbb{F} = \mathbb{F}_p$ for a prime p. There exists a linear-time randomized algorithm that takes as input a co-nondeterministic circuit D with m-bit inputs and a stride $0 \le s \le r-1$ and outputs either a nondeterministic or co-nondeterministic circuit P of the same size as D together with a bit b indicating whether it is nondeterministic $(b = 1)$ or co-nondeterministic $(b = 0)$ with the following guarantee: If D is a distinguisher for $\mathrm{SU}(\hat{g})$ with output length m, then with probability at least $\frac{2}{3m}$, P has advantage at least $\frac{1}{3m}$ at predicting the last bit of the seeded HSG H underlying the definition of $\mathrm{SU}(\hat{g})$ with fixed stride s.*

Proof. Let D be a co-nondeterministic circuit with m input bits that is a distinguisher for $\mathrm{SU}(\hat{g})$ (and in particular for the seeded HSG H). This means that

$$\Pr_{r \in \{0,1\}^m}[D(r) = 0] \le \frac{1}{2} \text{ and } \Pr_{(\vec{y},k,s) \in \mathbb{F}^r \times \{0,1\}^{\log n' + \log m}}[D\left(H(\vec{y}, j, s)\right) = 0] = 1.$$

We use D to obtain a predictor for the last bit of $H(y, j, s)$ given the previous elements for a fixed stride s. Consider randomly sampling $1 \le i \le m$ and $\sigma_1, \ldots, \sigma_m \in \{0, 1\}$, and constructing the circuit $D_i(g_1, \ldots, g_{i-1}) = D(g_1, \ldots, g_{i-1}, \sigma_i, \ldots, \sigma_m) \oplus \sigma_i$. Note, first, that the value of σ_i determines whether this circuit is nondeterministic or co-nondeterministic, since $\sigma_i = 1$ negates the output (resulting in a nondeterministic circuit), while $\sigma_i = 0$ does not (resulting in a co-nondeterministic circuit). Moreover, let X be a random variable describing the advantage that D_i has at predicting the i-th bit of $H(y, j, s)$ over random $y, k, s, m, \sigma_1, \ldots, \sigma_m$. By Yao's prediction versus indistinguishability result, we get that $\mathbb{E}[X] \ge \frac{1}{2m}$. By Markov's inequality, it follows that $\Pr\left[X \ge \frac{1}{3m}\right] \ge \frac{1}{3m}$.

Now, say we have a "good" predictor D_i for the i-th bit of H (with advantage at least $\frac{1}{3m}$). That is, with noticeable probability over $\vec{y}, j$, on input $E(\hat{g}(M^{p^s}\vec{y}))_j, E(\hat{g}(M^{2\cdot p^s}\vec{y}))_j, \ldots, E(\hat{g}(M^{(i-1)\cdot p^s}\vec{y}))_j$, D_i outputs $E(\hat{g}(M^{i\cdot p^s}\vec{y}))_j$. By feeding it $m-1$ inputs and having it ignore the first $m-i$ inputs, we obtain a circuit that, again with noticeable probability over $\vec{y}, j$, when given input $E(\hat{g}(M^{p^s}\vec{y}))_j, E(\hat{g}(M^{2\cdot p^s}\vec{y}))_j, \ldots, E(\hat{g}(M^{(m-1)\cdot p^s}\vec{y}))_j$, outputs $E(\hat{g}(M^{m\cdot p^s}\vec{y}))_j$, i.e., a predictor for the last bit of H with fixed stride s. Call this modified version P. The distinguisher-to-predictor algorithm outputs P and σ_i. ■

We remark that it is not necessary to know the actual circuit input to the distinguisher-to-predictor transformation, and black-box access to the underlying deterministic predicate suffices (in which case the procedure outputs a (co)-nondeterministic circuit with oracle access to the underlying predicate).

Now, we present a technical lemma that spells out the additional information required by the Shaltiel-Umans reconstruction and the success probability for the reconstruction. We also make two changes to the original result: First, we allow for multiple nondeterministic predictors per stride (instead of a single one) and second, we also make it explicit that the reconstruction works with additive error in the positive/negative prediction probabilities for each predictor. The reason for allowing multiple predictors is that this allows us to run the distinguisher-to-predictor transformation of Lemma 3.11 multiple times per stride s in the

hope that it produces one "good" predictor.

Lemma 3.12 (Following [SU05]). *Let $\hat{g} : \mathbb{F}^r \to \mathbb{F}$ be a polynomial of degree at most Δ, where $\mathbb{F} = \mathbb{F}_p$ for some prime p. Let also $m \leq p$ and r' be such that $p \geq (r+1)r'$. Finally, let H be the seeded version of the SU generator. There exists a nondeterministic algorithm A that receives the following as input:*

- *The generator matrix M used by the* SU *generator.*
- *A list of predictors $P_1, \ldots, P_k : \{0,1\}^{m-1} \to \{0,1\}$, each one given as a circuit of size m such that for every stride $0 \leq s \leq r-1$, there exists some predictor P_i that has advantage (over a random choice of y and j) at least $\frac{1}{3m}$ at predicting the last bit of $H(y,j,s)$. Also, bits $\sigma_1, \ldots, \sigma_m$ indicating if these predictors are nondeterministic ($\sigma_i = 1$) or co-nondeterministic ($\sigma_i = 0$). Formally, for each $0 \leq s \leq r-1$ there exists a predictor P_i such that*

$$\Pr_{\vec{y} \in \mathbb{F}^r, j \in [n']} [P_i(H(\vec{y},j,s)_1, H(\vec{y},j,s)_2, \ldots, H(\vec{y},j,s)_{m-1}) = H(\vec{y},j,s)_m] \geq \frac{1}{3m}$$

- *Values $\rho'_{1,0}, \rho'_{1,1} \ldots, \rho'_{1,r-1}, \rho'_{2,0}, \ldots, \rho'_{k,r-1}$ such that $\rho'_{i,s}$ is a $\left(\frac{1}{24m}\right)$-additive approximation for the probability $\rho_{i,s}$ (over random $\vec{y} \in \mathbb{F}^r$ and $j \in [n]'$) that P_i predicts σ_i for fixed stride s. Formally, we have $\rho_{i,s} - \frac{1}{24m} \leq \rho'_{i,s} \leq \rho_{i,s} + \frac{1}{24m}$ where*

$$\rho_{i,s} = \Pr_{\vec{y} \in \mathbb{F}^r, j \in [n']} [P_i(H(\vec{y},j,s)_1, H(\vec{y},j,s)_2, \ldots, H(\vec{y},j,s)_{m-1}) = \sigma_i]$$

  for all $i \in [k]$, $0 \leq s \leq r-1$.

- *$(r+1)r'$ points from $\mathbb{F}^r$*

$$\vec{y}_{0,1}, \vec{y}_{0,2}, \ldots, \vec{y}_{0,r'}, \vec{y}_{1,1}, \vec{y}_{1,2}, \ldots, \vec{y}_{1,r'}, \ldots, \vec{y}_{r,1}, \vec{y}_{r,2}, \ldots, \vec{y}_{r,r'}$$

  as well as $(r+1)r'$ distinct points from $\mathbb{F}$

$$t_{0,1}, t_{0,2}, \ldots, t_{0,r'}, t_{1,1}, t_{1,2}, \ldots, t_{1,r'}, \ldots, t_{r,1}, t_{r,2}, \ldots, t_{r,r'}$$

  These determine two curves C_1 and C_2 that are used throughout the reconstruction. Specifically,

- $C_1 : \mathbb{F} \to \mathbb{F}^r$ *is the degree* $\nu = (r+1)r' - 1$ *curve for which* $C_1(t_{i,j}) = \vec{y}_{i,j}$ *for all* i, j, *and*
- $C_2 : \mathbb{F} \to \mathbb{F}^r$ *is the degree* $\nu = (r+1)r' - 1$ *curve for which* $C_2(t_{1,j}) = \vec{y}_{1,j}$ *for all* j *and* $C_2(t_{i,j}) = A^{p^{i-2}}\vec{y}_{i,j}$ *for* $i \geq 2$ *and all* j.

- *The evaluation of* $\hat{g}$ *at* $\vec{0} \cup \bigcup_{i=1}^{m} M^i(C_1 \cup C_2)$.
- *A point* $\vec{x} \in \mathbb{F}^r$.

A runs in time $\mathsf{poly}(k, p, r, m)$ *and outputs a value* $v \in \mathbb{F}$. *If it holds that* $p \geq 32\Delta\nu\,(24m)^3$, *then with probability at least* $1 - \Theta\left(krp^r 2^{-r'}\right)$ *over random choices for the* $\vec{y}_{i,j}$*'s and* $t_{i,j}$*'s, it holds that* $v = \hat{g}(\vec{x})$ *for every input* $\vec{x} \in \mathbb{F}^r$. *Finally, A only requires oracle access to the deterministic predicates underlying each predictor.*

Before we sketch the proof of Lemma 3.12, we show how we use it to establish Lemma 3.10.

Proof of Lemma 3.10. First, we define the generator H_{det}. On input $z \in \{0,1\}^n$, H_{det} computes the low-degree extension $\hat{z}$ of z with parameters h, p and r to be defined later (though recall we need that $h^r \geq n$ and we will need to set p large enough by Lemma 3.12). Then, it outputs $\text{SU}(\hat{z})$ with output length m, where SU is the Shaltiel-Umans generator. This procedure runs in time $\mathsf{poly}(p^r, m)$ and outputs at most that many strings of length m.

Now, we define the A_{comp} algorithm. When given inputs 1^m and $z \in \{0,1\}^n$, A_{comp} outputs the following values:

1. The generator matrix M, which it computes using the procedure detailed in the beginning of this section.
2. The curves C_1 and C_2, randomly selected as in Lemma 3.12, and the necessary evaluations of $\hat{z}$ along the curves (the next-to-last bullet of Lemma 3.12).
3. The random bits required to run the uniform distinguisher-to-predictor transformation of Lemma 3.11 a total of k times, k/s times per stride s (for k to be defined later) to obtain a list of k predictors $P_1, \ldots, P_k$ (each with oracle to D).

4. The values required for computing the estimates $\rho'_{i,s}$ for each $1 \leq i \leq k$ and $0 \leq s \leq r-1$. To obtain these values, for each $1 \leq i \leq k$ and $0 \leq s \leq r-1$, A_{comp} randomly selects $\vec{y} \in \mathbb{F}^r$ and $j \in [n']$, computes $v = H(\vec{y}, j, s)$ using the matrix M and $\hat{z}$ and then outputs v. A_{comp} does this as many times as necessary to obtain the required approximation with "low" failure probability (that we discuss in more detail below).

The algorithm A_{dec}, on input the values output by $A_{\text{comp}}(1^m, z)$ and an index i' in $[n]$, performs the following actions:

1. Using the random bits output by A_{comp}, A_{dec} computes the distinguisher-to-predictor transformation of Lemma 3.11 a total of k times to obtain a list of k predictors $P_1, \ldots, P_k$ and the bits $\sigma_1, \ldots, \sigma_k$.

2. Computes the estimates $\rho'_{i,s}$ for each $1 \leq i \leq k$ and $0 \leq s \leq r-1$. To compute a value $\rho'_{i,s}$, it feeds the first $m-1$ bits of each value v output by A_{comp} into the predictor P_i with stride s, and queries the SAT oracle to determine the output of the predictor, finally setting $\rho'_{i,s}$ to the estimated probability that P_i with stride s predicts σ_i. As each sample/oracle query is independent of the others, these can all be carried out in parallel.

3. Finally, it queries its SAT oracle to determine the last bit of the output of algorithm A (note that A_{dec} has access to all of the inputs required by A) on the point $\vec{x}$ that corresponds to bit position i' of z.

Now, we set some values and argue the correctness, compression, efficiency and input access properties of $(A_{\text{comp}}, A_{\text{dec}})$.

Correctness. Assume that H_{det} does not output a hitting set for D. By Lemma 3.11, each application of the distinguisher-to-predictor transformation is successful with probability at least $1/3m$. By running it $3mk'$ times for each stride, a Chernoff bound guarantees that the probability that no execution is successful is at most $2^{k'/3}$ for sufficiently large k'. By a union

bound, the probability that we obtain at least one "good" predictor per stride is at least $1 - m \cdot 2^{k'/3}$, which can be made negligible (in m) by taking $k' = m$ and thus producing a total of $k = O(m^3)$ predictors. Similarly, we can make the probability of getting some estimate $\rho'_{i,s}$ with error more than $\frac{1}{24m}$ negligible in m by taking $\mathsf{poly}(m)$ samples per predictor/stride. Conditioned on the list of predictors and the estimates being "good", the probability that A with the computed values is correct is at least $1 - \Theta\left(krp^r 2^{-r'}\right) \geq 1 - \Theta\left(m^3 rp^r \cdot 2^{-r'}\right)$. A union bound then guarantees correctness with essentially the same probability as the additional negligible error probability gets absorbed by the $\Theta(\cdot)$. By setting $h = m$, $r = \log n / \log m$, $p = \Theta(m^6 \cdot \log^4 n)$ and $r' = \Theta(\log m + r \log p)$, we have that $32\Delta(r+1)r'(24m)^3 \leq m^5 \log^3 n$ and thus p is large enough for Lemma 3.12 to apply. Given our parameter choices, the probability that A computes $\hat{z}$ correctly, and thus $(A_{\text{comp}}, A_{\text{dec}})$ output z correctly, is at least $2/3$ for sufficiently large n and m.

Compression. We bound the length of each value output by A_{comp}. The matrix M has description length $\mathsf{poly}(r, \log p) = \mathsf{polylog}(m, n)$. The curves C_1 and C_2 are each described by $(r+1)r' = \mathsf{poly}(r, \log m, \log p)$ points, and thus also have description length $\mathsf{polylog}(m, n)$. The algorithm also outputs at most $m \cdot \mathsf{polylog}(m, n)$ evaluations of $\hat{z}$, and each evaluation has length $\log p = \mathsf{polylog}(m, n)$, resulting in length $\mathsf{poly}(m, \log n)$. Each distinguisher-to-predictor transformation requires $O(m)$ bits, and there are $k = \mathsf{poly}(m)$ of these, resulting in $\mathsf{poly}(m)$ bits total for those. Finally, the $\mathsf{poly}(m)$ samples per predictor/stride result in a total length of $\mathsf{poly}(m, \log n)$. Thus, in total, the output length of A_{comp} on input z of length n and 1^m is $\mathsf{poly}(m, \log n)$.

Efficiency. We start by bounding the running time of A_{comp}. Computing the matrix M takes time $\sqrt{p^r} \cdot \mathsf{poly}(p, r) \leq n \cdot \mathsf{poly}(m, \log n)$. Tossing the coins required to determine the curves C_1 and C_2 and to run the distinguisher-to-predictor reductions takes time $\mathsf{poly}(m, \log n)$. A_{comp} also computes $\mathsf{poly}(m, \log n)$ evaluations of $\hat{z}$ for items 2 and 4, and each such evaluation takes time $n \cdot \mathsf{poly}(h, \log p, r) = n \cdot \mathsf{poly}(m, \log n)$. Finally, computing each value of $H(\vec{y}, j, s)$ takes time $\mathsf{poly}(p, r, m) = \mathsf{poly}(m, \log n)$, and there are $\mathsf{poly}(m)$ of those. Taking everything

together, the running time of A_{comp} is upper bounded by $n \cdot \mathsf{poly}(m, \log n)$.

We now upper bound the running time of A_{dec}. Running the $\mathsf{poly}(m)$ many distinguisher-to-predictor transformations takes time $\mathsf{poly}(m)$, as does computing the estimates $\rho'_{i,s}$. Then, querying the SAT oracle on the output of A takes time $\mathsf{poly}(k, p, r, m) = \mathsf{poly}(m, \log n)$. Finally, we still need to use Fact 3.22 to allow for computing A_{dec} with only one round of non-adaptive SAT queries instead of two. Taking everything together, the final running time for A_{dec} is $\mathsf{poly}(m, \log n)$ for recovering one bit of z. ■

Now, we turn back to sketching the argument for Lemma 3.12. To compute $\hat{g}(\vec{y})$ on input $\vec{y}$, the original reconstruction also needs to know the values $0 \le d_y \le p^r - 1$ such that $\vec{y} = M^{d_y}\vec{1}$ and $0 \le a \le p^r - 1$ such that $C_1(1) = M^a\vec{1}$. In the original version of the SU reconstruction, the value of a is hardcoded into the circuit and they employ a non-standard low-degree extension over the original hard function f to allow for efficiently computing d_y for the $\vec{y}$ that represent an input of f. In the uniform deterministic setting, this is a significant challenge, which is overcome in [CLO+23] by combining the SU generator with a generator based on this discrete logarithm problem for M. In our case, we can use nondeterminism to guess-and-verify d_y and a whenever they are needed, which allows for computing $\hat{g}(\vec{y})$ for any $\vec{y} \in \mathbb{F}^r \setminus \{\vec{0}\}$ without significantly affecting the running time.

The original reconstruction [SU05, Theorem 6.5] employs a procedure called *Nondeterministic Learn Next Curve*, which learns the evaluations of $\hat{g}$ on a low-degree curve $C : \mathbb{F} \to \mathbb{F}^r$ given a small number of reference points for the values of $\hat{g} \circ C$ as well as some evaluations of $\hat{g}$ on which it runs a "good" predictor for the generator at a specific stride s. The procedure approximately computes the necessary predictions and uses error-correction to recover a list of polynomials, one of which is supposed to be $\hat{g} \circ C$ (assuming that the predictor is "good"). To determine which one is the correct polynomial, it evaluates each polynomial in the list and compares the results with the reference points. The only modification we make to the reconstruction is that we take the union of the lists of polynomials over all k predictors (since we don't know which one is "good" for stride s). With this change, we may obtain a

larger list of candidates for the correct polynomial (by a factor of k). A union bound over the introduced errors leads to the multiplicative factor of k in the error probability for the reconstruction. The remainder of the Shaltiel-Umans reconstruction uses the *Nondeterministic Learn Next Curve procedure* as a blackbox, and is thus unaffected by the change. For completeness, we also provide a full proof in the following section.

3.4.1 Proof of technical Lemma

This section is dedicated to proving Lemma 3.12, which we do by retracing the original proof due to Shaltiel and Umans [SU05] and making modifications as necessary. We need a standard tail inequality for ν-wise independent random variables.

Lemma 3.13 ([BR94]). *Let $\nu \geq 6$ be an even integer. Suppose $X_1, X_2, \ldots, X_n$ are ν-wise independent random variables taking values in $[0,1]$. Let $X = \sum_{i=1}^{n} X_i$ and $A > 0$. Then*

$$\Pr\left[|X - E[X]| \geq A\right] \leq 8 \cdot \left(\frac{\nu \cdot E[X] + \nu^2}{A^2}\right)^{\nu/2}.$$

We also need a well-known list-decoding algorithm due to Sudan.

Lemma 3.14 ([Sud97]). *Let $m, \alpha, \deg$ be integers. Given m many distinct pairs (x_i, y_i) from a field $\mathbb{F}$ with $\alpha > \sqrt{2 \cdot \delta \cdot m}$ there are at most $2m/\alpha$ polynomials g of degree δ such that $g(x_i) = y_i$ for at least α pairs. Furthermore, a list of all such polynomials can be computed in time $\mathsf{poly}(m, \log|\mathbb{F}|)$.*

We now dedicate the remainder of this section to the proof of Lemma 3.12.

The main difference between the non-uniform and uniform versions of the SU reconstruction is the *Nondeterministic Learn Next Curve* procedure, so we focus on it. To define it, though, we first need to compute some values related to the probability estimates $\rho'_{1,0}, \rho'_{1,1}, \ldots, \rho'_{1,r-1}, \rho'_{2,0}, \ldots, \rho'_{t,r-1}$. Recall $\rho_{i,s}$ is the probability that predictor P_i predicts the value σ_i (that is, the value that is easy to verify). Recall the guarantee for each $\rho'_{i,s}$

$$\rho_{i,s} - \frac{1}{24m} \leq \rho'_{i,s} \leq \rho_{i,s} + \frac{1}{24m}$$

for all i, s.

The algorithm A first computes the values

$$n_{i,s} = pn' \left(\rho'_{i,s} - \frac{1}{12m} \right)$$

for all i, s. Those values are used in the *Nondeterministic Learn Next Curve* procedure that we define next.

Nondeterministic Learn Next Curve. We define the procedure almost exactly as in the original construction, but with minor modifications to account for the fact that we have multiple predictors per stride s (at least one of which is "good"). The procedure receives the following as input:

- Next curve $C : \mathbb{F} \to \mathbb{F}^r$, a degree ν polynomial.
- Reference points $R \subseteq \mathbb{F}$, a set of r' distinct elements from $\mathbb{F}$.
- Stride s, an integer in $[0, \ldots, (r-1)]$.
- Input evaluations $\{a_t^i\}_{t \in \mathbb{F}, i \in [1,\ldots,(m-1)]}$ and $\{b_t\}_{t \in R}$, elements of $\mathbb{F}$ whose intended values are $a_t^i = \hat{g}(A^{-ip^s} C(t))$ and $b_t = \hat{g}(C(t))$.

The procedure outputs a sequence of values $\{c_t\}_{t \in \mathbb{F}}$, elements of $\mathbb{F}$ whose intended values are $c_t = \hat{g}(C(t))$. That is, it computes the values of $\hat{g}$ restricted to C. The procedure works as follows:

1. For each $1 \le i \le k$, guess a set T_i of $n_{i,s}$ distinct pairs $(t_j, z_j) \in \mathbb{F} \times [n']$ and a "witness" string w_j for each such pair.
2. Check that this is a "good guess", that is,

$$\forall (t_j, z_j) \in T_i, P_i \left(E(a_{t_j}^{m-1})_{z_j}, E(a_{t_j}^{m-2})_{z_j}, \ldots, E(a_{t_j}^{1})_{z_j}; w_j \right) = \sigma_i.$$

   If is it not, then reject.

3. For all $1 \le i \le k, t \in \mathbb{F}$ and $z \in [n']$, set $r_z^{i,t} = \sigma_i$ if $(t, z) \in T_i$ or $1 - \sigma_i$ otherwise.

4. For all $1 \le i \le k$ and $t \in \mathbb{F}$, set $S_{i,t}$ to be the list of γ^{-2} messages (where $\gamma = \frac{1}{24m}$) whose encodings differ from $r^{i,t}$ in at most $(1/2 - \gamma)n'$ places.

    - For each $1 \le i \le k$, apply Lemma 3.14 on the pairs $\{(t, e)\}_{t \in \mathbb{F}, e \in S_{i,t}}$ with $\alpha = \gamma p/4$ and $\delta = \Delta \cdot \nu$, and collect all polynomials into a single list.
    - If the list is empty, reject and output $\perp$. If the list contains a unique polynomial $f(t)$ for which $f(t) = b_t$ for all $t \in R$, output $\{f(t)\}_{t \in \mathbb{F}}$, otherwise reject and output $\perp$.

Fix a stride s. We say that *Nondeterministic Learn Next Curve* procedure is successful for stride s, next curve C and reference points R if, when receiving these as input together with the intended input evaluations, for every nondeterministic guess it either rejects or outputs the intended evaluations of $\hat{g}$. Moreover, it must be the case that there exists a "good guess" that it accepts. We now argue that over a random choice of curve C and reference points R, the procedure is successful with high probability as long as there exists a "good" predictor for stride s among the $P_1, \ldots, P_k$. The idea is to show that, with high probability over C, the fraction of points for which each P_i predicts σ_i is close to the ρ_i (and thus close to ρ_i'), and also that the fraction of points for which a "good" predictor is correct is close to the predictor's advantage. When both conditions hold, we are guaranteed to have enough agreement so that the correct polynomial f appears in the final list-decoding phase. Finally, we show that the probability (again over C) that some other polynomial passes the final verification phase (where we evaluate the polynomial on all reference points) is small, and thus the procedure outputs the correct evaluations of f.

Claim 3.15. *Assume that $p \ge 32\Delta\nu\,(24m)^3$ and that, for all strides $0 \le s \le r - 1$ there exists some predictor P_i that has advantage (over random y, j) at least $\frac{1}{3m}$ at predicting the last bit of $G'(y, j, s)$. Then, for all strides s, the probability over C and R that* Nondeterministic

Learn Next Curve *is successful for s, C and R is at least $1 - 3k \cdot 2^{-r'}$, where $C : \mathbb{F} \to \mathbb{F}^r$ is a uniformly chosen degree ν curve, and $R \subseteq \mathbb{F}$ is a uniformly chosen subset of $\mathbb{F}$ of size r'.*

Proof. Fix a stride s. For some $1 \le i \le k$, we first show that the fraction of points on which P_i predicts σ_i along C is close to the fraction of points on which P_i predicts σ_i on the whole space. Define the random variables

$$X_t = \left|\left\{z \in [n'] \mid P_i\left(E(a_t^{m-1})_z, E(a_t^{m-2})_z, \ldots, E(a_t^1)_z\right) = \sigma_i\right\}\right|$$

and let $X = \sum_{t \in \mathbb{F}} X_t$. Note that the X_t's are ν-wise independent random variables, that

$$X = \left|\left\{(t, z) : P_i\left(E(a_t^{m-1})_z, E(a_t^{m-2})_z, \ldots, E(a_t^1)_z\right) = \sigma_i\right\}\right|,$$

and that $E[X] = pn'\rho_{i,s}$. By Lemma 3.13, we have that

$$\Pr\left[|X - E[X]| \ge \frac{1}{24m} \cdot pn'\right] \le 8 \cdot \left(\frac{\nu \cdot pn' + \nu^2}{\left(\frac{1}{24m} \cdot pn'\right)^2}\right)^{\nu/2} \le 8 \cdot \left(\frac{2\nu}{\left(\frac{1}{24m}\right)^2 \cdot pn'}\right)^{\nu/2} \le 2^{-\nu/2},$$

where the last two inequalities hold because $p \ge 32\Delta\nu\,(24m)^3$. This means that with probability at least $1 - 2^{-\nu/2}$,

$$pn'\left(\rho_{i,s} - \frac{1}{24m}\right) \le X \le pn'\left(\rho_{i,s} + \frac{1}{24m}\right). \tag{3.1}$$

Define (3.1) as the first "good" event. In particular, we get that

$$X \ge pn'\left(\rho'_{i,s} - \frac{1}{12m}\right) = n_{i,s}. \tag{3.2}$$

Define the second "good" event as the event that (3.2) holds for all predictors $P_1, \ldots, P_k$. By a union bound, the second "good" event happens with probability at least $1 - k \cdot 2^{-\nu/2}$. If the second "good" event happens, then there exists at least one "good" guess for each T_i and the witnesses w_j. Moreover, due to the error in the approximation $\rho'_{i,s}$, the procedure guesses that between

$$pn'\left(\rho_{i,s} - \frac{1}{12m}\right) \text{ and } pn'\left(\rho_{i,s} + \frac{1}{12m}\right)$$

predictions from P_i evaluate to σ_i. This differs from the actual value (in the high-probability case we are considering) by a fraction of at most $\frac{1}{6m}$. Therefore, it holds that for all $1 \le i \le k$ and any such "good" guess

$$\left|\left\{(t,z) \mid r_z^{i,t} = P_i\left(E(a_t^{m-1})_z, E(a_t^{m-2})_z, \ldots, E(a_t^1)_z; s\right)\right\}\right| \ge \left(1 - \frac{1}{6m}\right) pn'.$$

Now, assume there exists some predictor among $P_1, \ldots, P_k$ that has "good" advantage at predicting the generator H for stride s. Say this is predictor P_i. We show that P_i is correct along the curve C on almost the same fraction of points as it is correct in the entire space. Define the random variable

$$Y_t = \left|\left\{z \in [n'] \mid P_i\left(E(a_t^{m-1})_z, E(a_t^{m-2})_z, \ldots, E(a_t^1)_z\right) = E(\hat{g}(C(t)))_z\right\}\right|,$$

and define $Y = \sum_{t \in \mathbb{F}} Y_t$. Again, notice that the Y_t's are ν-wise independent and that $E[Y] \ge pn'\left(\frac{1}{2} + \frac{1}{3m}\right)$ since P_i has "good" advantage. By Lemma 3.13, we have, similar to before, that

$$\Pr\left[|Y - E[Y]| \ge \frac{1}{12m} pn'\right] \le 2^{-\nu/2}.$$

By a union bound, the probability that the first two "good" events happen simultaneously is at least $1 - (k+1)2^{-\nu/2}$. In that case, we have that

$$\left|\left\{(t,z) \mid r_z^{i,t} = E(\hat{g}(C(t)))_z\right\}\right| \ge \left(\frac{1}{2} + \frac{1}{12m}\right) pn'.$$

By an averaging argument, we have that for at least a $\gamma = \frac{1}{24m}$ fraction of the t's,

$$\left|\left\{z \mid r_z^{i,t} = E(\hat{g}(C(t)))_z\right\}\right| \ge \left(\frac{1}{2} + \frac{1}{24m}\right) n'.$$

For these t, the relative Hamming distance between $r^{i,t}$ and $E(\hat{g}(C(t)))$ is at most $1/2 - \gamma$, so $S_{i,t}$ contains $\hat{g}(C(t))$ and thus the list of polynomials produced by the procedure contains the polynomial $f(t) = \hat{g}(C(t))$.

Now, still conditioned the first two "good" events happening, consider a nondeterministic choice for the procedure that passes the first verification (meaning it is a "good guess"). Each

application of Lemma 3.14 produces a list of at most $8\gamma^{-3}$ degree $\Delta\cdot\nu$ univariate polynomials $f(t)$ that contains all polynomials for which $f(t) \in S_{i,t}$ for at least $\gamma p/4$ values of t, and thus the procedure produces a list of at most $8k\gamma^{-3}$ polynomials. By the previous discussion, the list of polynomials produced by the procedure contains the polynomial $\hat{g}(C(t))$, so at least one polynomial passes the final verification where we compare the polynomials with the reference points, and this is true for every "good guess".

Now, to show that, with high probability, there exists a "good guess" that is not ultimately rejected by the procedure, we need to show that at least one "good guess" produces a list of polynomials such that no other polynomial in the list agrees with $\hat{g}(C(t))$ on all of the reference points. Note that, in that case, the procedure outputs the correct evaluations, since only the polynomial $f(t) = \hat{g}(C(t))$ passes the final verification. Recall that two polynomials of degree $\Delta\cdot\nu$ over $\mathbb{F}_p$ can agree on at most a $\frac{\Delta\cdot\nu}{p}$ fraction of their points. This means that, for a fixed "good guess", the probability that an incorrect polynomial from the list agrees with $\hat{g}(C(t))$ on r' random points is at most

$$(8k\gamma^{-3})\left(\frac{\Delta\cdot\nu}{p}\right)^{r'} \leq \left(8k(24m)^3\right)\left(\frac{\Delta\cdot\nu}{32\Delta\nu(24m)^3}\right)^{r'} \leq k\cdot 2^{-r'}.$$

Clearly, this also serves as an upper bound for the probability that all "good guesses" lead to a "bad" list of polynomials containing an incorrect polynomial that agrees with $\hat{g}(C(t))$ on r' random points. Let the third "good" event be the event where at least one "good guess" leads to the correct output.

Finally, note that as long as the three "good" events happen, then the procedure outputs $\hat{g}(C(t))$ for $t \in \mathbb{F}$ for *all* "good guesses", as well as rejects every "bad" nondeterministic guess. Moreover, there exists a "good guess". A union bound guarantees that all "good" events happen with probability at least

$$1 - (k+1)\cdot 2^{-\nu/2} + k\cdot 2^{-r'}.$$

Recall that $\nu = (r+1)r'$. Since $r+1 \geq 2$, it follows that this is at most

$$(k+1)\cdot 2^{-r'} + k\cdot 2^{-r'} \leq 3k\cdot 2^{-r'}.$$

■

Moreover, note that *Nondeterministic Learn Next Curve* runs in time $\mathsf{poly}(k, p, n', \gamma^{-1}, m)$ = $\mathsf{poly}(k, p, m)$ since $n' = \mathsf{poly}(\log p, \gamma^{-1})$ and $\gamma^{-1} = O(m)$. Shaltiel and Umans [SU05] show how to, on input $\vec{y} \in \mathbb{F}^r$, $0 \leq a \leq p^r - 1$ such that $C_1(1) = A^a\vec{1}$, $0 \leq d_y \leq p^r - 1$ such that $\vec{y} = A^{d_y}\vec{1}$ and the evaluation of $\hat{g}$ at $\bigcup_{i=1}^{m} A^i(C_1 \cup C_2)$, use $O(pmr)$ invocations of the *Nondeterministic Learn Next Curve* procedure along curves C_1 and C_2 to compute $\hat{g}(\vec{y})$. Since the reconstruction is already nondeterministic, we can guess and verify both a and d_y and then run their procedure, which results in a total running time of $\mathsf{poly}(k, p, r, m)$. They also show that, if *Nondeterministic Learn Next Curve* is successful for any triple s, C, R with probability at least τ over C_1, C_2, then the entire reconstruction is successful (in the sense that it outputs a nondeterministic circuit that computes $\hat{g}$ correctly at any point in $\mathbb{F}^r$) with probability at least

$$1 - p^{-r} - (2rp^r)\tau \geq 1 - (9krp^r)2^{-r'}$$

over C_1, C_2. This concludes the proof of the lemma.

3.5 Equivalence

In this section, we develop our main results, equivalences between mild derandomization of prAM and the existence of leakage-resilient hard functions/relations. We first state a more general version of Theorem 3.3 for classes $\mathcal{C}$ such that $\mathsf{P}^{\mathcal{C}} \subseteq \mathcal{C}$. For such classes, we obtain an equivalence with respect to a hard function.

Theorem 3.16. *Let $\mathcal{C}$ be a complexity class such that $\mathsf{P}^{\mathcal{C}} \subseteq \mathcal{C}$. There exists a constant c such that for all $\epsilon \in (0, 1)$, the following are equivalent.*

- $\mathsf{prAM} \subseteq \mathcal{C}$.
- *There exists a length-preserving function $f \in \mathcal{C}$ that is (n^c, n^ϵ)-leakage-resilient hard on almost-all inputs against $\mathsf{prBPP}_{||}^{\mathsf{SAT}}$.*

Theorem 3.16 follows from Theorem 3.18 in Section 3.5.1 and Theorem 3.23 in Section 3.5.2. Examples of classes for which Theorem 3.16 applies are P^{NP} and $\mathsf{ZPP}^{\mathsf{NP}}$ and their time-$2^{\mathsf{polylog}(n)}$ and time-$2^{n^{o(1)}}$ variants.

In the case of $\Sigma_2\mathsf{P}$, we state a more general equivalence with respect to a hard relation.

Theorem 3.17. *Let $\mathcal{T}$ be a time bound. There exists a constant c such that for all $\epsilon \in (0,1)$, the following are equivalent.*

- $\mathsf{prAM} \subseteq \bigcup_{k\in\mathbb{N}} \Sigma_2\mathsf{TIME}[n^k \cdot T(n^k)]$.
- *There exists $k \in \mathbb{N}$ and a total length-preserving relation $R \in \Sigma_2\mathsf{TIME}[n^k \cdot T(n^k)]$ that is (n^c, n^ϵ)-leakage-resilient hard on almost-all inputs against $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$.*

Theorem 3.17 follows from Theorem 3.19 in Section 3.5.1 and Theorem 3.25 in Section 3.5.2. Similar to Theorem 3.16, Theorem 3.17 applies to the time-$\mathsf{poly}(n)$, time-$2^{\mathsf{polylog}(n)}$ and time-$2^{n^{o(1)}}$ variants of $\Sigma_2\mathsf{P}$. Theorem 3.3 follows by instantiating Theorems 3.16 and 3.17 with polynomial time bounds.

3.5.1 From leakage-resilient hardness to derandomization

Assuming the existence of a leakage-resilient hard function/relation against $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$, we show that we can obtain mild derandomization of prAM. Our approach is to instantiate the Shaltiel-Umans generator with the value of $f(x)$ (or some $y \in R(x)$ in case of a hard relation R). The reconstructor for the algorithm can be described as a pair (Leak, A). In case the generator fails, $\mathrm{Leak}(x, f(x))$ outputs with high probability a small string π such that $A(x,\pi)$ recovers $f(x)$. Leak is a probabilistic algorithm and A is a $\mathsf{P}^{\mathsf{SAT}}$ algorithm.

We now prove the hardness-to-derandomization direction of Theorem 3.16.

Theorem 3.18. *Let $\mathcal{C}$ be a complexity class such that $\mathsf{P}^{\mathcal{C}} \subseteq \mathcal{C}$ and $\mathsf{NP} \subseteq \mathcal{C}$. There exists a constant $c \geq 1$ such that the following holds. Assume that there exist a constant $\epsilon \in (0,1)$*

and a length-preserving function $f \in \mathcal{C}$ that is (n^c, n^ϵ)-leakage-resilient hard on almost-all inputs against $\mathsf{prBPP}_{||}^{\mathsf{SAT}}$. Then there exists a targeted hitting-set generator H for co-nondeterministic circuits computable in $\mathcal{C}$, implying that $\mathsf{prAM} \subseteq \mathcal{C}$.

Proof. Fix a binary string representation for co-nondeterministic circuits that describes a circuit of size m by a string of length $m' = \Theta(m \log m)$. The generator H, on input a string x of length m' describing a co-nondeterministic circuit D_x of size m, sets $n = m^a$ for a sufficiently large constant a to be defined later and sets $x' = x0^{n-m'}$. It then computes $f(x')$ and instantiates the generator H_{det} of Lemma 3.10, outputting the set $S_x = H_{\text{det}}(1^m, f(x'))$. Computing $f(x')$ can be done in $\mathcal{C}$ by the assumption that $\mathsf{P}^{\mathcal{C}} \subseteq \mathcal{C}$ and since $|x'| = \mathsf{poly}(m)$, and computing S_x from $f(x')$ takes deterministic time $\mathsf{poly}(m, n) = \mathsf{poly}(m)$, and therefore the generator H is computable in $\mathcal{C}$.

Assume, with the intent of deriving a contradiction, that H fails as a targeted hitting-set generator for prAM. This means that there exists an infinite set $\mathcal{D}$ of co-nondeterministic circuits that accept at least a $1/2$ fraction of their inputs such that $H(D)$ fails to hit every $D \in \mathcal{D}$. We show that there exist probabilistic algorithms Leak and A with non-adaptive oracle access to SAT running in time n^c for a constant c to be defined later such that for infinitely many x', $\text{Leak}(x', f(x'))$ produces $|x'|^\epsilon$ bits of leakage and $A(x', \text{Leak}(x', f(x')))$ computes $f(x')$ with high probability.

The algorithm Leak parses the input $x' \in \{0,1\}^n$ as $x' = x0^{n-m'}$ for $m' = \Theta(m \log m)$ and $m = n^{1/a}$. If x' is not of the expected type, it outputs 0^n. Otherwise, it outputs $\pi = A_{\text{comp}}(f(x'), 1^m)$. The algorithm A, on input x' and π, similarly parses x' as $x0^{n-m'}$ and outputs $A_{\text{dec}}(\pi, D_x)$, where D_x is the circuit described by x.

Notice that we measure the running times of Leak and A in terms of $n = m^a$. By Lemma 3.10, Leak runs in time $\mathsf{poly}(m, n) = \mathsf{poly}(n) \leq n^k$ for a sufficiently large constant k. By the same lemma, n^k serves as an upper bound for the running time of A, which is $n \cdot \mathsf{poly}(m, \log n)$ to compute the entirety of $f(x')$. Moreover, the amount of leakage is $\mathsf{poly}(m, \log n) \leq (m \log n)^k$, by possibly redefining the value of k. By taking $a = 2k/\epsilon$, this is

at most n^ϵ.

As for correctness, Lemma 3.10 guarantees that for all x such that $H(D_x)$ fails to hit D_x, $A_{\mathrm{comp}}(1^m, f(x'))$ for $x' = x0^{n-m'}$ outputs with probability at least $2/3$ a string π such that $A_{\mathrm{dec}}(\pi, D_x)$ computes the mapping $i \mapsto f(x')_i$. In that case, A with inputs x' and π outputs $f(x')$. The conclusion $\mathsf{prAM} \subseteq \mathcal{C}$ follows as in Proposition 3.9: Given an Arthur-Merlin protocol P witnessing $\Pi \in \mathsf{prAM}$ and an input x, we construct $D_{P,x}$ and compute a hitting set $S = H(D_{P,x})$. Finally, we use the fact that $\mathsf{NP} \subseteq \mathcal{C}$ to verify in $\mathcal{C}$ that $D_{P,x}$ rejects all $\rho \in S$, rejecting otherwise. ∎

Essentially the same argument establishes the result for a leakage-resilient hard relation $R \in \Sigma_2\mathsf{TIME}[T(n)]$.

Theorem 3.19. *There exists a constant $c \geq 1$ such that the following holds. Assume that there exist a constant $\epsilon \in (0,1)$ and a length-preserving relation $R \in \Sigma_2\mathsf{TIME}[T(n)]$ that is (n^c, n^ϵ)-leakage-resilient hard on almost-all inputs against $\mathsf{prBPP}_{||}^{\mathsf{SAT}}$. Then there exists a targeted hitting-set generator for prAM computable in the class $\Sigma_2\mathsf{TIME}[T(\mathsf{poly}(m))]$, implying that $\mathsf{prAM} \subseteq \bigcup_{k\in\mathbb{N}} \Sigma_2\mathsf{TIME}[T(n^k)]$.*

Proof (sketch). The proof follows the argument of Theorem 3.18 closely. Instead of computing $H_{\mathrm{det}}(f(x'), 1^m)$, the generator H guesses and verifies $y \in R(x')$ and then outputs $H_{\mathrm{det}}(y, 1^m)$, which leads to a generator computable in time $T(m^a) + \mathsf{poly}(m) = T(\mathsf{poly}(m))$. The reconstruction is identical, and allows for computing $y \in R(x')$ given a small amount of leakage on y. ∎

We remark that the arguments for Theorems 3.18 and 3.19 show, in particular, that it is possible to compute $f(x')$ (or $y \in R(x')$) *locally* in time $\mathsf{poly}(m, \log n)$ by having A take additional input i and run $A_{\mathrm{dec}}(\pi, D_x, i)$.

3.5.2 From derandomization to leakage-resilient hardness

We first establish the derandomization-to-hardness direction of Theorem 3.16. As mentioned in Section 3.1, we frame the problem of computing a leakage-resilient hard function as a $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ search problem and then makes use of a search-to-decision reduction as in [Gol11].

At a high-level, a $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ search problem is a search problem R that admits two polynomial-time probabilistic algorithms with non-adaptive oracle access to SAT, G and V. V verifies that a candidate solution for R is correct with high probability, and captures the decisional version of the search problem. However, it is also necessary that a solution can be produced efficiently, and thus G outputs solutions for S with high probability. The search-to-decision reduction gives us a deterministic algorithm with oracle access to some problem in $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ that outputs solutions for R. Since $\mathsf{prBPP}_{\|}^{\mathsf{SAT}} \subseteq \mathsf{P}_{\|}^{\mathsf{prAM}}$, the oracle can be substituted by a $\mathsf{P}_{\|}^{\mathsf{prAM}}$ oracle. For a class $\mathcal{C} \in \{\mathsf{P}^{\mathsf{SAT}}, \mathsf{ZPP}^{\mathsf{SAT}}\}$, we can use the fact that $\mathsf{P}^{\mathcal{C}} \subseteq \mathcal{C}$ together with the derandomization assumption on prAM to compute the leakage-resilient hard function using the search-to-decision reduction in $\mathcal{C}$. For $\Sigma_2\mathsf{P}$, we need to be more careful, since it is unknown whether $\mathsf{P}^{\Sigma_2\mathsf{P}} \subseteq \Sigma_2\mathsf{P}$, and indeed the inclusion is believed to be false as otherwise the polynomial hierarchy collapses to $\Sigma_2\mathsf{P}$. In this case, we show that a Σ_2-derandomization of prAM implies a derandomization of the same strength for $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$, then have a Σ_2-algorithm guess and verify a solution for the search problem.

To carry out the strategy above, we start by defining $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ search problems.

Definition 3.20. Let R_Y and R_N be two disjoint binary relations. We say that (R_Y, R_N) is a $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ search problem if the following two conditions hold.

1. The decisional promise problem represented by (R_Y, R_N) is in $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$; that is, there exists a probabilistic polynomial-time algorithm V with non-adaptive oracle access to SAT such that for every $(x, y) \in R_Y$ it holds that $\Pr[V(x, y) = 1] \geq 2/3$ and for every $(x, y) \in R_N$ it holds that $\Pr[V(x, y) = 1] \leq 1/3$.

2. There exists a probabilistic polynomial-time algorithm G with non-adaptive oracle

access to SAT such that, for every x for which $R_Y(x) \neq \varnothing$, it holds that $\Pr[G(x) \in R_Y(x)] \geq 2/3$, where $R_Y(x) = \{y \mid (x,y) \in R_Y\}$.

◂

We observe that the search-to-decision strategy developed for prBPP-search problems in [Gol11] relativizes. Thus, to extend the result to $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$-search problems, it suffices to argue that if the search problem only requires non-adaptive oracle access to SAT, then the same holds for the oracle used in the search-to-decision reduction. Note, however, that the reduction itself is still adaptive, it just requires an oracle for a problem in $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$.

Proposition 3.21. *For every* $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ *search problem* (R_Y, R_N)*, there exists a binary relation* R *such that* $R_Y \subseteq R \subseteq (\{0,1\}^* \times \{0,1\}^*) \setminus R_N$ *and solving the search problem of* R *is (adaptively) deterministically polynomial-time reducible to some decisional problem in* $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$*.*

Proof of Proposition 3.21. The argument is essentially identical to that for Theorem 3.5 in [Gol11] (an analogue of Proposition 3.21 for prBPP search problems) with two modifications/observations. For completeness, we sketch the proof of Goldreich's construction: Let (R_Y, R_N) be a prBPP search problem, and let G be a solution-finding algorithm and V a verification algorithm for the problem. We define the algorithm $G'(x, r, \rho) = V(x, G(x; r); \rho)$, that is, G' takes input x and a random sequence r for G, which it uses to produce a candidate solution, and a random sequence ρ for V, which it uses to verify if this is a good solution. Given an input x, the strategy is to determine the bits of r one by one while maintaining the invariant that a random continuation r'' of the current prefix r' (which is initially empty) satisfies G' with high probability. This is where the decisional prBPP-oracle comes into play: It is used to verify the invariant by randomly sampling r'' and ρ, computing $G'(x, r'r'', \rho) \in \{0,1\}$ multiple times and checking whether the average is over a specific threshold. Once r is found, we can output $G(x; r)$ as a solution.

The first issue that needs to be addressed to port the argument to the $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ setting is that computing G' in the natural way requires two rounds of non-adaptive queries. Using Fact 3.22, stated next, it is possible to compute G' with a single round of non-adaptive queries at only a polynomial slowdown.

Fact 3.22 (See e.g., [SU06, Lemma 7.2]). *There exists a non-adaptive* SAT*-oracle algorithm* M *with the following behavior. On input* $x \in \{0,1\}^n$ *and the description of two non-adaptive* SAT*-oracle algorithms* M_1 *and* M_2 *such that* M_1 *runs in time time* $t_1(n)$ *and produces outputs of length* $t_1(n)$ *and* M_2 *takes inputs of length* $t_1(n)$ *and runs time* $t_2(n)$*,* M *runs in time* $\mathsf{poly}(n, t_1(n), t_2(n))$ *and outputs* $M_2(M_1(x))$*.*

The second modification is in fact an observation: Since computing G' can be done with non-adaptive oracle access to SAT, we are able to approximate the acceptance probability of G' over a random continuation r'' by running G' multiple times in parallel and comparing the average with the threshold. With these modifications, we guarantee that the decisional problem used as an oracle by the search procedure also only requires non-adaptive oracle access to SAT, i.e., is in $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$. ∎

We are now ready to prove the direction of derandomization to hardness of Theorem 3.16.

Theorem 3.23. *Let* $\mathcal{C}$ *be a complexity class such that* $\mathsf{P}^{\mathcal{C}} \subseteq \mathcal{C}$*. If* $\mathsf{prAM} \subseteq \mathcal{C}$*, then for all constants* c *and* $\epsilon \in (0,1)$ *there exists a length-preserving function* $f \in \mathcal{C}$ *that is* $(n^c, n - \omega(1))$*-leakage-resilient hard on almost-all inputs against* $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$*, where* $\omega(1)$ *is polynomial-time computable.*

Proof. Our approach is to cast computing a leakage-resilient hard function f as a $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ search problem, which allows us to instantiate Proposition 3.21 and show that there is a $\mathsf{P}^{\mathsf{prBPP}_{\|}^{\mathsf{SAT}}} \subseteq \mathsf{P}^{\mathsf{P}_{\|}^{\mathsf{prAM}}} \subseteq \mathsf{P}^{\mathsf{prAM}}$ algorithm that solves it. Finally, we use the derandomization assumption together with the assumption on $\mathcal{C}$ to conclude that f is computable in $\mathcal{C}$.

First, we argue that for any x, a random choice of $f(x)$ is hard w.r.t. a fixed pair (Leak, A) of probabilistic algorithms with non-adaptive oracle access to SAT. Fix a constant

c, an input $x \in \{0,1\}^n$ and a polynomial-time computable function $\nu(n) = \omega(1)$. Let ρ_{Leak} and ρ_A denote the random bits input to Leak and A, respectively. Let

$$P(r, \rho_{\text{Leak}}, \rho_A) \equiv |\text{Leak}(x, r; \rho_{\text{Leak}})| \leq \ell \wedge A(x, \text{Leak}(x, r; \rho_{\text{Leak}}); \rho_A) = r,$$

for $\ell = n - \nu(n)$. We say that r fails if $\Pr_{\rho_{\text{Leak}}, \rho_A}[P(r, \rho_{\text{Leak}}, \rho_A)] \geq 1/6$.

We have that

$$\begin{aligned}
\Pr_r[r \text{ fails}] &= \Pr_r[\Pr_{\rho_{\text{Leak}}, \rho_A}[P(r, \rho_{\text{Leak}}, \rho_A)] \geq 1/6] && \text{[definition of failing } r\text{]} \\
&\leq 6 \cdot \mathbb{E}_r[\Pr_{\rho_{\text{Leak}}, \rho_A}[P(r, \rho_{\text{Leak}}, \rho_A)] && \text{[Markov's inequality]} \\
&= 6 \cdot \mathbb{E}_{r, \rho_{\text{Leak}}, \rho_A}[P(r, \rho_{\text{Leak}}, \rho_A)] && \text{[expectation of indicator variable]} \\
&= 6 \cdot \mathbb{E}_{\rho_{\text{Leak}}, \rho_A}[\mathbb{E}_r[P(r, \rho_{\text{Leak}}, \rho_A)]] && \text{[reordering random bits]} \\
&< 6\frac{2^{\ell+1}}{2^n} && \text{[pigeonhole argument]}
\end{aligned}$$

The pigeonhole argument is that, after fixing the random bits ρ_{Leak} and ρ_A, for each of the at most $2^{\ell+1}$ strings y of length at most ℓ, among all the strings r that Leak maps to y, there is at most one that A maps back to r. Setting $\ell = n - \nu(n)$ implies that the probability that a string r fails is at most $12/2^{\nu(n)}$.

We then define the search problem (R_Y, R_N) such that $(x, r) \in R_Y$ if $|x| = |r| = n$ and for the first $\nu(n)$ pairs of probabilistic machines with non-adaptive oracle access to SAT (Leak, A) clocked to run in time n^c, it holds that

$$\Pr_{\rho_{\text{Leak}}, \rho_A}[|\text{Leak}(x, r; \rho_{\text{Leak}})| \leq \ell \wedge A(x, \text{Leak}(x, r); \rho_A) = r] < \frac{1}{6}. \tag{3.3}$$

As for "no" instances, $(x, r) \in R_N$ if for at least one pair out of the first $\nu(n)$ (Leak, A), equation (3.3) with $1/6$ replaced by $1/3$ does not hold.

Now, we show that this problem is a $\mathsf{prBPP}_{||}^{\mathsf{SAT}}$ search problem. On input $x \in \{0,1\}^n$, the algorithm for finding a solution samples a random $r \in \{0,1\}^n$. By a union bound over the $\nu(n)$ many algorithms Leak and A and the fact that r fails for a particular pair with probability at most $12/2^{\nu(n)}$, it holds that the solution-finding algorithm succeeds with probability at

least 2/3 for sufficiently large n. On input (x, r), the verification algorithm enumerates the first $\nu(n)$ probabilistic machines with non-adaptive oracle access to SAT, Leak and A, all clocked to run in time n^c. It then estimates the value

$$p_{\mathrm{Leak},A} = \Pr_{\rho_{\mathrm{Leak}},\rho_A}[|\mathrm{Leak}(x, r; \rho_{\mathrm{Leak}})| \le \ell \wedge A(x, \mathrm{Leak}(x, r); \rho_A) = r]$$

up to error 1/12 and with failure probability at most $1/n$ for each pair (Leak, A). By a standard Chernoff bound, it suffices to perform the following steps a polynomial (in n) number of times per pair: Let Leak' be the algorithm that, on input x and a random sequence ρ_{Leak} for Leak, computes an output $y = \mathrm{Leak}(x, r; \rho_{\mathrm{Leak}})$ and outputs (x, y). Let A' be an algorithm that, on input x, y and random sequence ρ_A, rejects if $|y| > \ell$ and outputs $A(x, y; \rho_A)$ otherwise, and let M be the algorithm of Fact 3.22. Select random strings ρ_{Leak} and ρ_A for Leak and A, respectively, compute $A'(x, \mathrm{Leak}'(x, r; \rho_{\mathrm{Leak}}); \rho_A)$ by feeding M inputs $x, r, \rho_{\mathrm{Leak}}, \rho_A$ and the codes of Leak' and A' and compare the output with r. Each such execution requires time $\mathsf{poly}(n)$ for a total running time of $\mathsf{poly}(n)$. Moreover, all oracle queries made by M can be made in parallel.

After estimating the average acceptance probability for each pair (Leak, A), the verification algorithm outputs 1 if the estimated values are less than 1/4 (the midpoint between 1/6 and 1/3) for all pairs of algorithms. By a union bound over the $\nu(n)$ pairs of algorithms, the algorithm accepts $(x, r) \in R_Y$ and rejects $(x, r) \in R_N$ with probability at least 2/3.

Finally, we use Proposition 3.21 together with the derandomization assumption and the assumption on $\mathcal{C}$ to conclude that there is a function $f \in \mathcal{C}$ that solves (R_Y, R_N), i.e., f is a leakage-resilient hard function. ∎

The proof of Theorem 3.23 shows that, given x, verifying whether a candidate r for $R(x)$ is hard w.r.t. the first $\nu(n)$ algorithms Leak, A can be done in $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$. The proof also shows that a "good" r always exists for sufficiently large n. To extend Theorem 3.23 to obtain a hard relation $R \in \Sigma_2$, we first show that a Σ_2-derandomization assumption on prAM implies the same derandomization for $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$.

Proposition 3.24. *Let T be a time bound. If* $\mathsf{prAMTIME}[n] \subseteq \Sigma_2\mathsf{TIME}[T(n)]$*, then it follows that* $\mathsf{prBPP}_{\|}^{\mathsf{SAT}} \subseteq \bigcup_{k\in\mathbb{N}} \Sigma_2\mathsf{TIME}[(n^k \cdot T(n^k)]$.

Proof. Let $\Pi \in \mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ and recall that $\mathsf{prBPP}_{\|}^{\mathsf{SAT}} \subseteq \mathsf{P}_{\|}^{\mathsf{prAM}}$ (Lemma 2.11). Let $L \in \mathsf{P}_{\|}^{\mathsf{prAM}}$ be a language that agrees with Π and M be a polynomial-time non-adaptive prAM-oracle algorithm that decides L. Without loss of generality, we may assume that M has non-adaptive oracle access to a complete problem Γ for prAM, and that $\Gamma \in \mathsf{prAMTIME}[n]$. Recall that $\mathsf{prAM} \subseteq \Pi_2\mathsf{P} = \mathsf{co}\Sigma_2\mathsf{P}$ and that, by assumption, $\mathsf{prAMTIME}[n] \subseteq \Sigma_2\mathsf{TIME}[T(n)]$. Let $L_\Gamma \in \Sigma_2\mathsf{TIME}[T(n)]$ be such that L_Γ agrees with positive instances of Γ (that is, $x \in \Gamma_Y \iff x \in L_\Gamma$) and $L_{\overline{\Gamma}} \in \Sigma_2\mathsf{P}$ be such that it agrees with negative instances of Γ (that is, $x \in \Gamma_N \iff x \in L_{\overline{\Gamma}}$). To obtain a Σ_2-simulation of M on input x, we first guess which of the queries M makes are answered positively, which are answered negatively, and which queries are outside of the promise of Γ. Then we verify the positive and negative queries using either the $\Sigma_2\mathsf{TIME}[T(n)]$ algorithm for L_Γ, if the query was guessed to be positive, or the $\Sigma_2\mathsf{P}$ algorithm for $L_{\overline{\Gamma}}$, if the query was guessed to be negative. In parallel, verify (using universal quantification) that M on input x accepts with the guessed positive/negative answers and any answer to the queries outside of the promise.

Since M is guaranteed to be correct with any answer to queries outside of the promise, if $x \in L$ then there exists an existential guess that leads the simulation to accept, namely the one that guesses each query type correctly. If $x \notin L$, no potentially accepting path can guess that some positive query is answered negatively or vice-versa, and thus the only way a mistake could happen is by incorrectly guessing that a set Q of positive/negative queries are outside of the promise. However, since acceptance of the simulation implies that the algorithm accepts with any answer to the queries in Q, it in particular implies that it accepts with the correct answers, a contradiction. This simulation runs in $\Sigma_2\mathsf{TIME}[n^k \cdot T(n^k)]$ for some constant k, which depends on the running time for M. ∎

Under a Σ_2-derandomization assumption for prAM, Proposition 3.24 allows a Σ_2 algo-

rithm to compute a leakage-resilient hard relation R by guessing a candidate y and verifying in Σ_2 that it is a "good" solution. We therefore establish Theorem 3.25.

Theorem 3.25. *Let T be a time bound. If $\mathsf{prAMTIME}[n] \subseteq \Sigma_2\mathsf{TIME}[T(n)]$, then for every constant c there exists a constant k and a total length-preserving relation $R \in \Sigma_2\mathsf{TIME}[n^k \cdot T(n^k)]$ that is $(n^c, n - \omega(1))$-leakage-resilient hard on almost-all inputs against $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$, where $\omega(1)$ is polynomial-time computable.*

3.5.3 Targeted hitting-set generators from derandomization

In this section, we show that mild derandomization of prAM implies the existence of targeted hitting-set generators that suffice to obtain the original derandomization result. The results follow as consequences of the equivalence between leakage-resilient hardness and derandomization since we show the hardness-to-derandomization direction by constructing such targeted generators.

Corollary 3.26. *Let $\mathcal{C}$ be a complexity class such that $\mathsf{P}^{\mathcal{C}} \subseteq \mathcal{C}$. If $\mathsf{prAM} \subseteq \mathcal{C}$, then there exists a targeted hitting-set generator for prAM computable in $\mathcal{C}$.*

Proof. By Theorem 3.18, there is a constant c such that if there is a length-preserving function $f \in \mathcal{C}$ that is $(n^c, n^{1/2})$-leakage-resilient hard on almost-all inputs against $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$, then there is a targeted hitting-set generator as in the conclusion. By Theorem 3.23, the assumption $\mathsf{prAM} \subseteq \mathcal{C}$ implies the existence of f as required. ∎

Corollary 3.27 is established in the exact same way.

Corollary 3.27. *Let T be a time bound. If $\mathsf{prAM} \subseteq \bigcup_{k\in\mathbb{N}} \Sigma_2\mathsf{TIME}[n^k \cdot T(n^k)]$, then there exists a constant k' and a targeted hitting-set generator for prAM computable in $\Sigma_2\mathsf{TIME}[n^{k'} \cdot T(n^{k'})]$.*

Theorem 3.5 follows by combining Corollaries 3.26 and 3.27 with polynomial time bounds.

3.5.4 Derandomization from hardness against learn-and-evaluate protocols

In this section, we prove that the hardness-to-derandomization direction of Theorems 3.16 and 3.17 holds under hardness against learn-and-evaluate protocols (as defined in Section 2.3.3). By combining such a result with the derandomization-to-hardness direction of Theorems 3.16 and 3.17, we obtain an equivalence in the leakage-resilient hardness setting between hardness against learn-and-evaluate protocols and hardness against $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$.

We define (T, ℓ)-leakage-resilient hardness on almost-all inputs against learn-and-evaluate protocols in a completely analogous way to hardness against $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ (Definition 3.1), where a probabilistic algorithm $\mathrm{Learn}(x, y)$ (which we take to be A^y_{learn}) takes on the role of Leak, running in time T and producing, on input x of length n, an output of length at most $\ell(n)$, and P_{eval} takes on the role of algorithm A, also running in time T.

We make use of the following lemma, which hinges on the RMV generator (Lemma 2.23).

Lemma 3.28. *There exists a deterministic algorithm H_{det}, a probabilistic algorithm* Learn *and an Arthur-Merlin protocol P_{eval} such that at least one of the following holds for every $z \in \{0,1\}^*$, $m \in \mathbb{N}$ and co-nondeterministic circuit D of size m that accepts at least half of its inputs:*

1. *$H_{det}(z, 1^m)$ outputs a set that hits D.*
2. *$P_{eval}(\mathrm{Learn}(z, 1^m), D)$ computes z with completeness 1 and soundness $2/3$.*

The construction also has the following properties:

- Compression: *On input z of length n and 1^m, the string output by A_{learn} has length $\mathsf{poly}(m, \log n)$.*
- Efficiency: *On input z of length n and 1^m, both H_{det} and A_{learn} run in time $\mathsf{poly}(m, n)$. P_{eval}, given a sketch π, D of size m and an additional index i, computes the i-th bit of z in time $(m \cdot \log n)^{O(\log^2 r)}$ for $r = O(\log n / \log m)$.*

- Input access: *The only way A_{dec} requires access to D is via blackbox access to the deterministic predicate that underlies D.*

Before proving Lemma 3.28, we remark that, while the running time for its evaluator P_{eval} does not scale as well as the construction based on SU (Lemma 3.10), it achieves the same compression length.

Proof of Lemma 3.28. First, we define the generator H_{det}. On input $z \in \{0,1\}^n$, H_{det} computes the low-degree extension $\hat{z}$ for z with parameters $h = m^{100}$, r the smallest power of two such that $h^r \geq n$ and p the smallest prime between Δ^{100} and $2\Delta^{100}$, where $\Delta = h \cdot r$. Then it outputs $\text{RMV}(\hat{z})$ with output length m. This procedure runs in time $\mathsf{poly}(m, p^r) = \mathsf{poly}(m, n)$ and outputs at most that many strings of length m.

We now define Learn and P_{eval}. Learn first computes the low-degree extension $\hat{z}$ exactly as H_{det}, and then simulates the honest commitment protocol as described in Lemma 2.23 using $\hat{z}$ to answer the learning queries posited by Arthur. P_{eval} is essentially identical to the protocol with the same name in Lemma 2.23, with the only difference being that on input i, it computes the corresponding element $\vec{z} \in \mathbb{F}^r$ before executing the protocol from Lemma 2.23.

By setting $s = \Theta(1/n^2)$, it holds that with probability at least $1 - 1/n^2$, $\text{Learn}(z, 1^m)$ outputs a sketch π such that $P^D_{\text{eval}}(\pi, \cdot)$ computes the mapping $i \mapsto z_i$ and soundness $1/n^2$, and thus computes z with soundness $2/3$. If D is not hit by $H_{\text{det}}(z, 1^m)$, then it also holds that $\text{Learn}(z, 1^m)$ outputs with probability 1 a sketch such that $P^D_{\text{eval}}(\pi, \cdot)$ computes z with completeness 1. Finally, Learn runs in time $\mathsf{poly}(n, \delta, r) = \mathsf{poly}(m, n)$ and produces a sketch π of length at most $\mathsf{poly}(m, \log n)$, and P_{eval} runs in time $\Delta^{O(\log^2 r)} \cdot \mathsf{polylog}(n) = (m \cdot \log n)^{O(\log^2 r)}$ to compute an individual bit of i. ∎

We now state the hardness-to-derandomization result that we obtain from Lemma 3.28.

Theorem 3.29. *Theorems 3.18 and 3.19 continue to hold when $\mathsf{prBPP}^{\mathsf{SAT}}_{\|}$ is replaced by learn-and-evaluate protocols.*

Proof. The proof follows closely that of Theorems 3.18 and 3.19 but uses the generator H_{det} of Lemma 3.28 instead of Lemma 3.10. For the reconstructor's running time, since we pick $n = \mathsf{poly}(m)$, we can upper bound the running time of Leak (which equals the algorithm Learn of Lemma 3.28) and the leakage-receiving protocol (which invokes P_{eval} n times) by n^c for some constant c. The bound on the leakage is calculated in the exact same way. ■

As a consequence, we obtain an equivalence between hardness on almost-all inputs against polynomial-time learn-and-evaluate protocols and leakage-resilient hardness on almost-all inputs against $\mathsf{prBPP}_{||}^{\mathsf{SAT}}$. For simplicity, we state the result for classes $\mathcal{C}$ such that $\mathsf{P}^{\mathcal{C}} \subseteq \mathcal{C}$, but it holds for Σ_2 as well.

Corollary 3.30. *Let $\mathcal{C}$ be a complexity class such that $\mathsf{P}^{\mathcal{C}} \subseteq \mathcal{C}$ and $\mathsf{NP} \subseteq \mathcal{C}$. There exists a constant c such that the following are equivalent for all $\epsilon \in (0,1)$:*

1. *There exists a length-preserving function $f \in \mathcal{C}$ that is (n^c, n^ϵ)-leakage-resilient hard on almost-all inputs against learn-and-evaluate protocols.*

2. *There exists a length-preserving function $f \in \mathcal{C}$ that is (n^c, n^ϵ)-leakage-resilient hard on almost-all inputs against $\mathsf{prBPP}_{||}^{\mathsf{SAT}}$.*

Proof. We start with the $1 \implies 2$ implication. If 1 holds, then by Theorem 3.29 it follows that $\mathsf{prAM} \subseteq \mathcal{C}$. This in turn implies 2 by Theorem 3.16. As for the other direction, if a function f is not (n^c, n^ϵ)-leakage-resilient hard on almost-all inputs against learn-and-evaluate protocols, then it is also not (n^c, n^ϵ)-leakage-resilient hard against $\mathsf{prBPP}_{||}^{\mathsf{SAT}}$ since a $\mathsf{prBPP}_{||}^{\mathsf{SAT}}$ algorithm can use the SAT oracle to determine the output of the evaluation protocol. ■

3.5.5 A direct construction of targeted generators from derandomization

In this section, we establish the implication of derandomization to targeted generators of Theorem 3.5 directly, without going through leakage-resilience as in Section 3.5.3. Moreover, we show that it is possible to obtain targeted pseudorandom generators for non-adaptive SAT-oracle circuits. These objects don't follow directly from the existence of a leakage-resilient hard relation/function and Theorems 3.18 and 3.19, since the generators obtained this way are hitting-set generators for co-nondeterministic circuits.

For completeness and because our focus has been on targeted hitting-set generators, we provide a formal definition for targeted pseudorandom generators. First, we define the notion of δ-fooling a distribution.

Definition 3.31. Let $\delta \in [0,1)$ be a constant $D : \{0,1\}^m \to \{0,1\}$ be a distribution, and S a multi-set of binary strings of length m. We say that S δ-fools D if the following holds:

$$\left| \Pr_{r \in \{0,1\}^m}[D(r) = 1] - \Pr_{s \in S}[D(s) = 1] \right| \leq \delta.$$

◂

We then extend Definition 3.8 to targeted pseudorandom generators for arbitrary classes of circuits.

Definition 3.32. Let $\mathcal{T}$ be a class of circuits and G be an algorithm that computes a relation $R \in \mathcal{C}\mathsf{TIME}[T(m)]$ between circuits of type $\mathcal{T}$ of size m and multi-sets of strings of length m. We say that G is a targeted pseudorandom generator for $\mathcal{T}$ computable in $\mathcal{C}\mathsf{TIME}[T(m)]$ if the following two conditions hold for all sufficiently large $m \in \mathbb{N}$ and some constant $\delta < 1/6$:

- For all $D \in \mathcal{T}$ of size m, there exists a non-empty S such that $(D, S) \in R$.
- For all $D \in \mathcal{T}$ of size m, every $S \in R(D)$ δ-fools D.

◂

Our approach is similar to the one in Section3.5.2: We view the process of computing a pseudorandom (multi-)set as a $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ search problem and then employ a search-to-decision reduction as in [Gol11]. For simplicity, we state the result for polynomial time bounds, but it applies to other time bounds such as quasipolynomial and subexpential as well.

Theorem 3.33. *Let $\mathcal{C} \in \{\mathsf{P}^{\mathsf{NP}}, \mathsf{ZPP}^{\mathsf{NP}}, \Sigma_2\mathsf{P}\}$. If $\mathsf{prAM} \subseteq \mathcal{C}$, then there exists a targeted pseudorandom generator for non-adaptive SAT-oracle circuits computable in $\mathcal{C}$.*

Proof. We cast computing a targeted pseudorandom set for non-adaptive SAT-oracle circuits as a $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ search problem, which allows us to instantiate Proposition 3.21, and then use the derandomization assumption to compute the pseudorandom set in the desired class.

The instances for the search problem (R_Y, R_N) consist of pairs (D, S) where D is a non-adaptive SAT-oracle circuit and S is a multi-set containing a constant number c of strings in $\{0,1\}^m$, for c to be defined later. We then define:

- $(D, S) \in R_Y$ if and only if $\left|\Pr_{r\in\{0,1\}^m}[D(r) = 1] - \Pr_{s\in S}[D(s) = 1]\right| < 1/7$.
- $(D, S) \in R_N$ if and only if $\left|\Pr_{r\in\{0,1\}^m}[D(r) = 1] - \Pr_{s\in S}[D(s) = 1]\right| \geq 1/6$.

Due to the gap between the two cases, a probabilistic algorithm with non-adaptive oracle access to SAT can solve the decisional problem (R_Y, R_N) in polynomial time and with high probability by approximating the acceptance probability of D up to small constant error $\epsilon = 0.01$ and checking whether the difference between the estimate and the probability that D accepts a random $s \in S$ is at most $1/7+\epsilon$. To compute a solution, it suffices to observe that for a suitable constant c, a random multi-set $S \subseteq \{0,1\}^m$ containing c strings approximates, with high probability, the acceptance probability of D up to error $1/7$.

The result now follows along the same lines as the end of the proofs of Theorems 3.23 and 3.25, where we use the derandomization assumption together with the search-to-decision reduction to derandomize the process of obtaining a solution (for deterministic and zero-error

probabilistic algorithms with oracle access to SAT) and use the derandomization assumption together with Proposition 3.24 to guess-and-verify a solution for Σ_2 algorithms.

■

As a Corollary, under the derandomization assumption of Theorem 3.33, we obtain a targeted *hitting-set* generator for non-adaptive SAT-oracle circuits that outputs a single string. This holds because the generator itself can check for acceptance with non-adaptive SAT oracle queries, which can be simulated by the three classes considered.

Corollary 3.34. *Let $\mathcal{C} \in \{\mathsf{P}^{\mathsf{NP}}, \mathsf{ZPP}^{\mathsf{NP}}, \Sigma_2\mathsf{P}\}$. If $\mathsf{prAM} \subseteq \mathcal{C}$, then there exists a targeted pseudorandom generator for non-adaptive SAT-oracle circuits computable in $\mathcal{C}$ that always outputs a single string.*

3.6 Connection to non-uniform lower bounds

In this section, we prove Theorem 3.7, which establishes further equivalences between white-box and blackbox mild derandomization of prAM. In particular, we show that uniform leakage-resilient hardness assumptions imply the $\Sigma_2\mathsf{E} \not\subseteq \mathsf{NP/poly}$ separation.

We need the equivalence between lower bounds for $\Sigma_2\mathsf{E}$ against nondeterministic circuits and against deterministic circuits with non-adaptive oracle access to SAT. Such an equivalence is known for classes $\mathcal{C}$ that admit low-degree extensions, in the sense that a boolean version of the low-degree extension of a language in $\mathcal{C}$ is also in $\mathcal{C}$ [SU06]. However, for a class to admit low-degree extensions, it needs to be closed under composition, such as E and $\mathsf{NE} \cap \mathsf{coNE}$, and is is unknown whether $\Sigma_2\mathsf{E}$ is closed under composition (and indeed this is believed to be false). Still, a standard argument sidesteps this issue and obtains the equivalence for $\Sigma_2\mathsf{E}$ as well. Since we only need this equivalence in the case of polynomial-size circuit lower bounds, we state it in this setting.

Lemma 3.35 (Instantiation of [SU06, Theorem 3.2]). $\Sigma_2\mathsf{E} \subseteq \mathsf{NP/poly}$ *if and only if* $\Sigma_2\mathsf{E} \subseteq \mathsf{P}_{\|}^{\mathsf{SAT}}/\mathsf{poly}$.

Proof. The direction $\Sigma_2\mathsf{E} \subseteq \mathsf{NP/poly} \implies \Sigma_2\mathsf{E} \subseteq \mathsf{P}_{\|}^{\mathsf{SAT}}/\mathsf{poly}$ is trivial, so we focus on the converse implication. Theorem 3.2 in [SU06] shows that if a low-degree extension $\hat{g}$ with specific parameters of a function g has non-adaptive SAT-oracle circuits of size s, then g itself has nondeterministic circuits of size $\mathsf{poly}(s)$. Assume that $\Sigma_2\mathsf{E} \subset \mathsf{P}_{\|}^{\mathsf{SAT}}/\mathsf{poly}$. By a standard argument, $\Sigma_2\mathsf{E}/n \subseteq \mathsf{P}_{\|}^{\mathsf{SAT}}/\mathsf{poly}$ and there exists a constant c such that every language in $\Sigma_2\mathsf{E}/n$ has non-adaptive SAT-oracle circuits of size n^c.

Let $L \in \Sigma_2\mathsf{E}$. By having as advice n bits describing the number of strings of length n in L, a Σ_2-algorithm can compute the characteristic function $g_L : \{0,1\}^n \to \{0,1\}$ of L by guessing which strings of length n are in L and verifying each one in $\Sigma_2\mathsf{E}$, outputting 1 if and only if the input string is in the list of guessed-and-verified strings. We then set parameters exactly as in [SU06], that is, for a parameter $r' = 2(n + \log(32n^{5c}))$, we set $h = (4r')^2(9n^c)^4$, $r = n/\log h + 3$ and p to the smallest prime greater than or equal to $9hdr'$ to obtain the low-degree extension $\hat{g}$ of g. With these parameters, it follows that the function $\hat{g}_{\mathrm{bool}}$ that maps the binary representation of a $\vec{y} \in \mathbb{F}_p^r$ and an index $i \in [\log p]$ to the i-th bit of $\hat{g}(\vec{y})$ is in $\Sigma_2\mathsf{E}/n$. Together with the assumption on $\Sigma_2\mathsf{E}$, we have that $\hat{g}$ has non-adaptive SAT-oracle circuits of size $s(n) = O(n^{c+1})$, and thus Theorem 3.2 in [SU06] guarantees that g (and thus L) has nondeterministic circuits of size $\mathsf{poly}(s(n)) = \mathsf{poly}(n)$. ∎

Now, we provide some intuition for the proof of Theorem 3.7. The first step is to understand how lower bounds such as $\Sigma_2\mathsf{E} \notin \mathsf{NP/poly}$ imply leakage-resilient hardness. First, by Lemma 3.35, the assumption implies also that $\Sigma_2\mathsf{E} \notin \mathsf{P}_{\|}^{\mathsf{SAT}}/\mathsf{poly}$. Define a function f that maps every input x of length ℓ to the truth-table of a hard language $L \in \Sigma_2\mathsf{E}$. With ℓ bits of advice (indicating how many strings of length ℓ are in L), this function can be computed in Σ_2-time $2^{O(\ell)}$. Assume, with the intent of deriving a contradiction, that f is not locally leakage-resilient hard on almost-all inputs against $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$. For each input x

where hardness fails, there exists a small leakage string a such that $A(x, a)$ locally computes f. That is, $L \in \mathsf{BPP}_{\|}^{\mathsf{SAT}}/\mathsf{poly}$ and thus also in $\mathsf{P}_{\|}^{\mathsf{SAT}}/\mathsf{poly}$ by Adleman's argument [Adl78]. As for the other direction, by using the techniques developed in Section 3.5, we show that leakage-resilient hardness is sufficient for obtaining the mild derandomization of item 1, and thus equivalent to non-uniform lower bounds at the low end by [AvM17].

We now detail how we prove Theorem 3.7. It is already known that items 1 and 2 are equivalent by the main result of [AvM17]. We show that 2 $\implies$ 3 in Lemma 3.36, that 3 $\implies$ 1 in Lemma 3.37 and that 1 $\iff$ 4 in Lemma 3.39.

We start by formalizing how to obtain leakage-resilient hardness from the assumption $\Sigma_2\mathsf{E} \notin \mathsf{NP}/\mathsf{poly}$.

Lemma 3.36. *Assume* $\Sigma_2\mathsf{E} \notin \mathsf{NP}/\mathsf{poly}$. *Then for all* $\epsilon > 0$ *there exists a relation* $R \in \Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$ *that is* $\mathsf{poly}(n)$*-local* $(\infty, \mathsf{poly}(n))$*-leakage-resilient hard on all inputs of infinitely-many input lengths against* $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$.

The conclusion of Lemma 3.36 should be interpreted in the following way: For any constant c, any pair (Leak, A) consisting of a function Leak with output length n^c and algorithm A than runs in local time n^c can only succeed at computing f on some input of finitely-many input lengths n.

Proof. Let $L \in \Sigma_2\mathsf{E}$ and fix $\epsilon > 0$. We construct a function f from L such that, if f is not hard as in the theorem statement, then $L \in \mathsf{NP}/\mathsf{poly}$. The existence of a hard relation then follows. Define f as the function that maps any input $x \in \{0,1\}^n$ to the truth-table of L at input length $n^{\epsilon/2}$. By having as advice the number N of strings of length $n^{\epsilon/2}$ in L, it is possible to compute this function in Σ_2-time $2^{O(n^{\epsilon/2})} \leq 2^{n^\epsilon}$ for sufficiently large n by guessing which N of the $2^{n^{\epsilon/2}}$ strings of length $n^{\epsilon/2}$ are in L and verifying those using the linear-exponential time Σ_2-algorithm for L. Note that since $f(x)$ is constant for all inputs of a given length, the same advice string applies to all such inputs.

Now, assume that f is not $\mathsf{poly}(n)$-local $(\infty, \mathsf{poly}(n))$-leakage-resilient hard on all inputs of infinitely-many input lengths against $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$. This means there exist a constant c, a (potentially uncomputable) function Leak and a probabilistic algorithm A with non-adaptive oracle access to SAT such that for almost-all input lengths n there exists $x \in \{0,1\}^n$ such that $|\mathrm{Leak}(x, f(x))| \leq n^c$ and $A(x, \mathrm{Leak}(x, f(x)))$ computes $f(x)$ locally in time n^c. By providing x and a "good" leakage string a as advice, algorithm $A(x, a)$ computes f locally in time n^c, and thus computes L at input length $n^{\epsilon/2}$ in time $\mathsf{poly}(n)$. This implies that $L \in \mathsf{BPP}_{\|}^{\mathsf{SAT}}/\mathsf{poly}$ and thus $L \in \mathsf{P}_{\|}^{\mathsf{SAT}}/\mathsf{poly}$ [Adl78]. By Lemma 3.35, $L \in \mathsf{NP}/\mathsf{poly}$. ∎

Now, we show that a slightly weaker hardness assumption, where the leakage-providing function is computable in subexponential time, suffices to derandomize prAM.

Lemma 3.37. *If for all $\epsilon > 0$ there exists a relation $R \in \Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$ that is $\mathsf{poly}(n)$-local $(2^{n^\epsilon}, \mathsf{poly}(n))$-leakage-resilient hard on all inputs of infinitely-many input lengths against $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$, then $\mathsf{prAM} \subseteq \text{io-}\Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$ for all $\epsilon > 0$.*

Proof. The proof is almost identical to that of Theorem 3.19, though padding the input is not necessary. Fix some $\epsilon > 0$, the idea to construct a targeted hitting-set generator is to, on input $x \in \{0,1\}^{m'}$ representing a circuit D_x of size m for $m' = \Theta(m \log m)$, guess-and-verify a value $y \in R(x)$ and instantiate the generator H_{det} of Lemma 3.10 with y and m. The process takes time $2^{O(m^\epsilon)}$ and requires $O(m^\epsilon)$ bits of advice (for computing $y \in R(x)$). As with Theorem 3.19, there exist probabilistic algorithms Leak and A with non-adaptive oracle access to SAT such that for any $x \in \{0,1\}^n$ representing a circuit D_x not hit by the generator there exists $y \in R(x)$ for which the following holds. On input (x, y), Leak runs in time $2^{O(n^\epsilon)}$ and produces $\mathsf{poly}(n)$ bits of leakage that, when given as input to A, allow it to locally compute y in time $\mathsf{poly}(n, \log(2^{n^\epsilon})) = \mathsf{poly}(n)$.

As the hardness assumption holds for every input of infinitely-many input lengths (and R is total on the same input lengths), the targeted generator also works for infinitely-many

circuit sizes m, and thus the derandomization in the conclusion of the theorem follows along the lines of Proposition 3.9. ∎

Finally, we show that the weak derandomization assumption of item 1 in Theorem 3.7 is equivalent to local leakage-resilient hardness against almost-maximal leakage. Before doing so, we present a variant of Proposition 3.24 for derandomizations with advice that only work for infinitely-many input lengths.

Proposition 3.38. *Assume that* $\mathsf{prAM} \subseteq \text{io-}\Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$ *for all* $\epsilon > 0$. *Then* $\mathsf{prBPP}_{\|}^{\mathsf{SAT}} \subseteq \text{io-}\Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$ *for all* $\epsilon > 0$.

Proof. Lemma 2.39 shows that $\Sigma_2\mathsf{E} \notin \mathsf{NP/poly}$ implies that $\mathsf{prBPP}_{\|}^{\mathsf{SAT}} \subseteq \text{io-}\Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$ for all $\epsilon > 0$. As the premise $\mathsf{prAM} \subseteq \text{io-}\Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$ implies that $\Sigma_2\mathsf{E} \notin \mathsf{NP/poly}$, we are done. ∎

Lemma 3.39. *The following are equivalent:*

1. $\mathsf{prAM} \subseteq \text{io-}\Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$ *for all* $\epsilon > 0$.

5. *For all* $\epsilon > 0$ *and* $c \geq 1$ *there exists a length-preserving relation* $R \in \Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$ *that is* n^c*-local* $(n^c, \ell(n))$*-leakage resilient hard on all inputs of infinitely-many input lengths against* $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$, *where* $n^{\Omega(1)} \leq \ell(n) \leq \omega(1)$ *is polynomial-time computable.*

Proof. The proof follows closely the arguments of Theorems 3.25 (in the derandomization-to-hardness direction) and 3.19 (in the hardness-to-derandomization direction).

In the derandomization-to-hardness direction, recall that the proof of Theorem 3.25 shows that for every constant c, checking whether a value y is $(n^c, n - \omega(1))$ leakage-resilient hard for a string x (w.r.t. the first few leakage-providing algorithms/attackers) can be done in $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$. Such a value y must also be n^{c-2}-local leakage-resilient hard. Otherwise, we could amplify the algorithm A that locally computes y so that it outputs every bit of y correctly with high probability and obtain an algorithm that computes y in its entirety in time n^c. Fix some $\epsilon > 0$. By the derandomization assumption and Proposition 3.38, the check can be

replaced by a $\Sigma_2\mathsf{TIME}[2^{n^\epsilon}]$ algorithm with n^ϵ bits of advice that is guaranteed to work on infinitely-many input lengths. By guessing a value of y and running the Σ_2 verification, we obtain a relation R that is defined and hard for all inputs of infinitely-many input lengths.

The hardness-to-derandomization direction follows Theorem 3.19 even more closely. Theorem 3.19 establishes the hardness-to-targeted-derandomization connection on an instance-wise basis, i.e., the targeted generator hits every co-nondeterministic circuit D_x described by a padded string x' for which the hardness assumption holds. Since hardness holds for all inputs of infinitely-many input lengths, we obtain a targeted generator that works for infinitely many circuit sizes, which as in Lemma 3.37 suffices to obtain the derandomization result. ■

Due to the gap between the length of a solution $y \in R(x)$ and the leakage-resilient hardness in item 3, together with the sub-optimal scaling of the RMV reconstructor (Lemma 3.28) when compared to the SU reconstructor (Lemma 3.10), this result does not extend to hardness against learn-and-evaluate protocols. Still, even though RMV requires more time for *decompression*, the learning part for both the RMV and the SU reconstructor achieve the same level of *compression* for the string used as a basis, where we view the sketch produced by the learning phase as a compressed representation of the input.

To conclude this section, we employ Corollary 3.34 to obtain a simple alternative proof of the result of [AGH+11] that $\mathsf{prAM} \subseteq \mathsf{P}^{\mathsf{NP}}$ implies E^{NP} has maximum circuit complexity.

Theorem 3.40 ([AGH+11]). *If* $\mathsf{prAM} \subseteq \mathsf{P}^{\mathsf{NP}}$*, then for all* $\epsilon \in (0,1)$ *there is a language in* E^{NP} *that requires circuits of size* $(1-\epsilon)\cdot 2^n/n$ *for all but finitely many input lengths* n*.*

Proof. Assume that $\mathsf{prAM} \subseteq \mathsf{P}^{\mathsf{NP}}$ and let H be the targeted hitting-set generator given by Corollary 3.34. Fix a constant $\epsilon \in (0,1)$. We define the language $L_\epsilon \in \mathsf{E}^{\mathsf{NP}}$ that has high circuit complexity. On input $x \in \{0,1\}^n$, we construct the SAT-oracle circuit C that, on input a string Y of length $N = 2^n$ representing the truth-table of a Boolean function on n input bits, accepts if and only if the circuit complexity of Y is at least $(1-\epsilon)\cdot 2^n/n$. C has

size $\mathsf{poly}(N)$, requires only a single SAT oracle query, and accepts at least half of the strings of length N by a standard counting argument. We then use H to obtain a string Y' that is accepted by C, and accept x if and only if the x-th bit of Y' equals 1. By construction, L has circuit complexity $(1-\epsilon)\cdot 2^n/n$, and the time required to construct C, compute $H(C)$ and then accept or reject an input x of length n is $\mathsf{poly}(N) = 2^{O(n)}$ with access to a SAT oracle. ■

3.7 Further research

We observe that the there is a common technique underlying the results for this chapter and the previous one: That of compression. In Chapter 2, we use failure of a PCP as a basis for the RMV generator to compress the PCP and obtain a speedup via the PCP verifier. This works because the proof length for a PCP system is the main bottleneck for the Merlin-Arthur protocol that it induces. The Chen-Tell result based on hardness on almost-all inputs is similar: Using the Nisan-Wigderson generator, they compress the layer polynomials for the circuit computing the hard function f, which leads to the speedup for computing f. In this chapter, we use the SU or RMV reconstructor to compress the value of $f(x)$ itself where f is a leakage-resilient hard function. In fact, focusing on compression instead of the speed-up seems to lead more easily to equivalences with derandomization, since a random string is incompressible with high probability. In the next chapter, we explore how this observation, together with a change in perspective, allows us to obtain further equivalences for derandomizing Arthur-Merlin protocols.

Chapter 4

Refuting Bottleneck Protocols

4.1 Introduction

This chapter focuses on our more recent contributions, which generalize many of the results of Chapters 2 and 3. A common theme throughout the chapter is the notion of compression and how it is useful for obtaining equivalences with derandomization. To introduce the ideas involved in our results, we start with a brief review of the recent results for derandomizing BPP and of our earlier results in the AM setting (Chapters 2 and 3).

BPP setting. As we have seen previously, whitebox derandomization for prBPP implies the existence of targeted pseudorandom generators that recover the derandomization result. 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 [Kor22a] 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]

Theorem 4.1 ([CTW23]). *The following are equivalent:*

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

For a class $\mathcal{A}$ of algorithms, $\mathcal{A}\mathsf{TICOMP}[t(n), \gamma(n)]$ denotes the class of computational processes obtained by first running a probabilistic algorithm A_{comp} and then an algorithm $A_{\text{dec}} \in \mathcal{A}$ on the output of A_{comp} such that both A_{comp} and A_{dec} run in time $t(n)$ and A_{comp} outputs a string of length at most $\gamma(n)$. Assuming $\gamma(n) < n$, 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 $(A_{\text{comp}}, A_{\text{dec}})$ as a *bottleneck* algorithm. A *refuter* for a function f against

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

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 3 in Theorem 4.1 is a relaxation of item 2. Note also that the existence of the refuter in item 2 or 3 only guarantees that, for any fixed $A' = (A_{\text{comp}}, A_{\text{dec}})$ in $\mathsf{prBPTICOMP}[t(n)^{1+\epsilon}, n^\epsilon]$, 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. In Chapter 2 we established a near-equivalence between whitebox derandomization for AM and 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 $\mathsf{prAM} \subseteq \mathsf{NP}$ (Theorem 2.5). In the other direction, assuming $\mathsf{prAM} \subseteq \mathsf{NP}$, 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 (Proposition 2.28). 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.

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]. In Chapter 2, 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 (Theorem 2.7). We also showed, in Chapter 3, that targeted hitting-set generators are

necessary for *mild* derandomization of prAM, i.e., simulations on NP-oracle or Σ_2 machines (Theorem 3.5). Along the way, we obtained an equivalence between mild derandomization of prAM and hardness in the presence of efficiently-computable leakage (Theorem 3.3).

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

Theorem 4.2. *The following are equivalent:*

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

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

3. *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* $\mathsf{prAMTICOMP}[n^{a+\epsilon}, n^{\epsilon}]$ *protocols with promised soundness for* f*.*

Consider item 2 in Theorem 4.2. Because of the bottleneck, any protocol $(A_{\mathrm{comp}}, P_{\mathrm{dec}})$ 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 4.2 shows that for derandomizing prAM, it suffices to derandomize trivial refuters for the identity function. In fact, as the relaxation in item 3 states, it suffices to derandomize trivial refuters for any function computable in NP.

Theorem 4.2 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 $\mathsf{poly}(T(\mathsf{poly}(n)))$. Similarly, a targeted hitting-set generator that runs in time T, and thus achieves the derandomization $\mathsf{prAM} \subseteq \mathsf{NTIME}[T(\mathsf{poly}(n))]$, results in a refuter for identity that runs in time $T(\mathsf{poly}(n))$. 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.

In Chapter 3 we showed that *mild* derandomization for prAM implies the existence of targeted hitting-set generators achieving the same derandomization result. Taken together with Theorem 4.2, we obtain an equivalence between mild derandomization and refutation. As with the equivalence of Theorem 4.2, the equivalence of Theorem 4.2 scales smoothly in terms of the running time for the refuter.

Theorem 4.3. *Let $\mathcal{C} \in \{\mathsf{P}^{\mathsf{NP}}, \mathsf{ZPP}^{\mathsf{NP}}, \Sigma_2\mathsf{P}\}$. The following are equivalent:*

1. $\mathsf{prAM} \subseteq \mathcal{C}$.
2. *There exists a targeted hitting-set generator for* prAM *computable in* $\mathcal{C}$.
3. *For some constant $\epsilon \in (0,1)$, there exists a refuter computable in $\mathcal{C}$ for the identity function against* $\mathsf{prAMTICOMP}[n^{1+\epsilon}, n^{\epsilon}]$.
4. *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 refuter computable in $\mathcal{C}$ against* $\mathsf{prAMTICOMP}[n^{a+\epsilon}, n^{\epsilon}]$.

It is worth comparing Theorem 4.3 with Theorem 3.3 and its extension to learn-and-evaluate protocols (Theorem 3.29). To do so, we recall the discussion in the introduction that relates refutation to leakage-resilient hardness. Fix a learn-and-evaluate protocol $P = (P_{\text{learn}}, P_{\text{eval}})$ with compression length $\gamma(n) < n$, and an input $x \in \{0,1\}^n$. Let P_x be P with fixed input x. Finding a value of $f(x)$ such that $P_x(f(x))$ fails to compute $f(x)$ is a refutation task for the identity function. The connection also holds in the other direction: a leakage-resilient hard function f gives us a refuter for the identity function that just outputs $f(x)$. Thus, there is an equivalence between leakage-resilient hard functions/relations and refuters for the identity function. It follows that Theorem 4.3 generalizes the main result of Chapter 3 by establishing an equivalence between mild derandomization and the existence of refuters for any function $f \in \mathsf{NP}$ instead of only for the identity function.

Also in the mild setting, we connect refutation to the known equivalence between white-box and blackbox derandomization of prAM at the low end. As seen in Chapter 3, the simulation $\mathsf{prAM} \subseteq \text{io-}\Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$ for all $\epsilon > 0$ is equivalent to the non-uniform lower bound $\Sigma_2\mathsf{E} \not\subseteq \mathsf{NP/poly}$ [AvM17]. We show that those conditions are also equivalent to refuters against non-adaptive SAT-oracle bottleneck algorithms with *polylogarithmic* compression length and with polylogarithmic-time *local* decompression algorithms, meaning the decompressor A_{dec} needs to produce each individual bit of the function in polylogarithmic time. Here is an informal version of the result:

Theorem 4.4 (informal). *The following are equivalent.*

1. *For all* $\epsilon > 0$, $\mathsf{prAM} \subseteq \text{io-}\Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$.
2. $\Sigma_2\mathsf{EXP} \not\subseteq \mathsf{NP/poly}$.
3. *There exists a low-end io-refuter* R *for the identity function against* $\mathsf{polylog}(n)$*-local* $\mathsf{prBPTICOMP}[n \cdot \mathsf{polylog}(n), \mathsf{polylog}(n)]^{\mathsf{SAT}}_{\|}$ *that is computable in* $\Sigma_2\mathsf{P}$ *with logarithmic advice.*

We refer the reader to Section 4.5.3 for the definition of low-end refuters, and for details on how to quantify the subexponential and polylogarithmic time and compression bounds. As the equivalence between derandomization and nondeterministic circuit lower bounds of [AvM17] is only known to apply to the low end of the derandomization spectrum, we only know Theorem 4.4 for this parameter setting. However, the implications $2 \implies 3$ and $3 \implies 1$ hold also for the high end and for derandomizations that work almost-everywhere.

The targeted hitting-set generators in Theorem 4.2 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 4.2 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 hitting-set generators 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 pseudorandom generators for prBPP.

Theorem 4.5. *If there exists a targeted hitting-set generator for* prAM *computable in nondeterministic time* $T(m)$*, then there exists a pseudorandom generator for* prBPP *computable in nondeterministic time* $\mathsf{poly}(T(\mathsf{poly}(m)))$*.*

Assuming a positive resolution to Goldreich's question, Theorem 4.5 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 [AGH+11].

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 4.6. *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 $\mathsf{poly}(T(\mathsf{poly}(n)))$ *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 4.6 were obtained under non-uniform hardness conditions for E in [KvM02; SU06] and under a slight variation of the uniform hardness assumption $\mathsf{E} \notin \text{io-}\mathsf{AMTIME}[2^{\epsilon n}]$ for some $\epsilon > 0$ in [GST03]. By Theorem 4.2 and its extensions, we can obtain such explicit constructions under the refutation hypothesis of Theorem 4.2.

Techniques. To establish the direction from refutation to derandomization of the main result for this chapter, we refine the instance-wise transformation of hardness into targeted hitting sets from Chapter 2. The approach in Chapter 2 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 Chapter 2 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 faster 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. However, the refuter is only guaranteed to work against bottleneck protocols, and the reconstructor from Theorem 2.24 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 $\mathsf{P}^{\mathsf{prAM}}$ 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 new reconstructor retains the *resilient soundness* property of Theorem 2.24. 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 4.6), including truth-tables with high circuit complexity.

Organization. In Section 4.2, we develop the ideas behind our main result for this chapter and relate them to existing techniques. We start the formal treatment in Section 4.3 with definitions, notation, and other preliminaries. In Section 4.4, we present the details of our

main result. We develop our results in the mild setting in Section 4.5. In Section 4.6, we discuss explicit constructions: Proposition 4.6 and examples of conditional explicit constructions, including hard truth-tables, rigid matrices, and Theorem 4.5.

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 4.2.

BPP setting. The known hardness vs. randomness tradeoffs are based on a pseudorandom generator construction G that takes a function h and outputs a pseudorandom 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 pseudorandom 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 *pseudorandom* 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) *pseudorandom* 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

$$A_{\mathrm{rec}}(h_x, D) \neq h_x \tag{4.1}$$

for at least one h_x, where $A_{\mathrm{rec}}(h_x, D)$ 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 (4.1) means that the reconstruction fails for h_x, and whenever that happens the targeted pseudorandom generator based on h_x has to fool D. Prior approaches all guarantee (4.1) 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 $A_{\mathrm{rec}}(h_x, D) = h_x$, 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 (4.1) 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 $\mathsf{prBPP} \subseteq \mathsf{P}$, 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 pseudorandom generator in the BPP setting.

Korten [Kor22a] 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 $A_{\mathrm{rec}}(z, D)$, 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 pseudorandom.

AM setting. In Chapter 2, 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 $\mathsf{polylog}(T(n))$ 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 [GST03] 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.

We saw in Chapter 2 that abstracting out the details of our construction and how the distinguisher D is obtained, we saw in Chapter 2 that 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. The pair (H, P_{rec}) respects the following property:

Property 4.7 (Property 2.10, restated). *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 4.7 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 2.10 does not hold). In that case, $H(z, D)$ must output a hitting set for D (the first item in Property 2.10 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 $\mathsf{prAM} \subseteq \mathsf{NP}$ from the existence of a refuter against bottleneck protocols with subpolynomial compression that run in time $n^{a+\epsilon} \cdot \mathsf{poly}(n)$ for some $\epsilon > 0$, where the $\mathsf{poly}(n)$ term comes from the use of PCPs.

We can do better, and get rid of the multiplicative $\mathsf{poly}(n)$ 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 $\mathsf{polylog}(n)$ instead of $\mathsf{poly}(n)$. 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 $n \cdot \mathsf{polylog}(n)$, 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 3 in Theorem 4.2 implies item 1.

Targeted generators to refutation. For the implication from item 1 to item 2 in Theorem 4.2, 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.

4.3 Preliminaries

In this section, we present preliminary definitions and results that are necessary for developing our contributions. We start by defining bottleneck algorithms and refuters, and then define PCPPs. The results in the chapter also rely on the preliminaries from the previous chapters.

4.3.1 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 4.8. Let $\mathcal{A}$ be a class of promise algorithms, t a time bound and $\gamma : \mathbb{N} \to \mathbb{N}$. We let $\mathcal{A}\mathsf{TICOMP}[t(n), \gamma(n)]$ be the class of computational problems with the following properties for some probabilistic algorithm A_{comp} and some $A_{\text{dec}} \in \mathcal{A}$: For any input $x \in \{0,1\}^*$:

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

◂

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. The differences do matter for the compression bound $\gamma(n)$. Constant-factor compression is not possible in general, and hard-wiring is not an option as it requires access to the full input.

Definition 4.8 applies to promise Arthur-Merlin protocols that output values, yielding the bottleneck protocol classes $\mathsf{prAMTICOMP}[t(n), s(n)]$. In the completeness and soundness notions of Definition 2.12, 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).

4.3.2 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 4.9. Let $f : \{0,1\}^* \to \{0,1\}^*$ be a total function, and $\mathcal{A}$ a resource-bounded semantic class of algorithms. A list-refuter for f against $\mathcal{A}$ is a meta-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 4.9 the behavior of a refuter 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 $\mathsf{prAMTICOMP}[t(n), \gamma(n)]$. 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 $\gamma(\ell)$, 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 $\mathsf{prAMTICOMP}[t(n), \gamma(n)]$ 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.

4.3.3 PCPs of proximity and error-correcting codes

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 4.10 (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 $L \in \mathsf{PCPP}_{s,\delta}[r,q,t]$ 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 4.11 ([BGH+05]). *Let T be a time bound and L be a pair language in the class $\mathsf{NTIME}[T(m,n)]$, 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 $L \in \mathsf{PCPP}_{s,\delta}[r,q,t]$, for*

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

- *Verification time* $t(m,n) = \mathsf{poly}(m, \log n, \log T(m,n))$.

We also use the following PCPP with a *canonical* proof strategy due to Paradise. Such a PCPP has a polynomial-time computable proof strategy that maps pairs of inputs in a pair language L to a proof. The disadvantage for this result is that it requires the pair language L to be computable deterministically.

Lemma 4.12 ([Par21]). *Let T be a time bound and L be a pair language in the class $\mathsf{DTIME}[T(m,n)]$, 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 $L \in \mathsf{PCPP}_{s,\delta}[r,q,t]$, for*

- *Proximity parameter* $\delta = 0.1$.
- *Randomness complexity* $r(m,n) = O(\log T(m,n))$.
- *Query complexity* $O(1)$.
- *Proof length* $\mathsf{poly}(T(m,n))$.
- *Verification time* $t(m,n) = \mathsf{poly}(m, \log n, \log T(m,n))$.

Moreover, there exists a polynomial-time computable proof strategy that maps $(x,z) \in L$ to y such that $\Pr[V^{z,y}(x,n) = 1] = 1$ and $\Pr[V^{z,y'}(x,n) = 1] < 1$ for every $y' \neq y$.

In our applications of the above PCPPs 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 $n \cdot \mathsf{polylog}(n)$ (e.g., see [Jus76] and the discussion in [Spi96]).

4.4 Equivalence

In this section, we establish Theorem 4.2. First, in Section 4.4.1, we present the refinement for the targeted hitting-set generator of Theorem 2.24. In Section 4.4.2, we prove the direction of refutation to targeted hitting-set generators. In Section 4.4.3, 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.

4.4.1 Targeted generator construction

We develop our targeted HSG as a refinement of Theorem 4.2. First, we describe the modifications needed to the construction and proof for the result. For completeness, we also include a full proof after discussing the modifications.

Theorem 4.13. *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 string output by A_{comp} has length $\mathsf{poly}(m, \log T(n))$.*
- Resilient soundness: *In both cases 1 and 2 above, the probability that the bottleneck protocol $P_{dec}(D, A_{comp}(1^m, z))$ 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 $n \cdot \mathsf{poly}(m, \log T(n))$. On inputs z of length n and D of size m, H runs in time $\mathsf{poly}(T(n), m)$ 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 $|f(z)| \cdot (m \cdot \log T(n))^{O((\log r)^2)}$.*

- Input access: *$H(x, D)$ only depends on x 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.*

We first describe the differences with relation to the proof of Theorem 2.24:

- 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 4.3.3. Note, in particular, that L_f is computable in nondeterministic time $n \cdot \mathsf{polylog}(n) + T(n)$. Let V be the PCPP verifier for L_f given by Lemma 4.11, 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}. The compresso 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 $\mathsf{poly}(m, \log T(n))$. 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 to which Merlin committed. 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 $(A_{\text{comp}}, P_{\text{dec}})$ 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 $\mathsf{poly}(m, \log T(n))$. For this reason, the running time for our reconstructor is dominated by the running time for the RMV reconstructor.

We now present a full proof for the result.

Proof of Theorem 4.13. 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 $n \cdot \mathsf{polylog}(n)$ as in Section 4.3.3. 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 $n \cdot \mathsf{polylog}(n) + T(n)$ 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 4.11 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 $\mathsf{poly}(T(n), n) = \mathsf{poly}(T(n))$ 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 $\mathsf{poly}(T(n))$. 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 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 $h^r = \mathsf{poly}(T(n), m)$ and $r = O(\log(T(n))/\log m)$. Throughout the rest of the proof, we denote by $\hat{y}_i$ the low-degree extension of each y_i with parameters p, h and r.

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

Generator. The generator H, on input z and a co-nondeterministic circuit D of size m, computes $\tilde{z} = \mathrm{Enc}(z)$ and the low-degree extension $\hat{z}$ of $\tilde{z}$ with the parameters above, and guesses the value of $v = f(z)$ and a PCPP y_i witnessing $(\tilde{z}, n, i, v_i) \in L_f$ for each index i of v. Then H verifies that $\Pr[V^{\tilde{z},y_i}(n,i,v_i) = 1] = 1$ for every $i \in [|v|]$ by deterministically enumerating the $\mathsf{poly}(T(n), m)$ random-bit strings for V. If any of the verifications fail, H fails. Otherwise, H computes the low-degree extension $\hat{y}_i$ of y_i. Finally, H outputs the union of $\mathrm{RMV}(\hat{z})$ and $\cup_{i \in [|v|]} \mathrm{RMV}(\hat{y}_i)$, where each invocation of the RMV generator is instantiated with the same output length m. Note that the choice of parameters for encoding $\hat{z}$ and each $\hat{y}_i$ respects the preconditions of Lemma 2.23.

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

Reconstructor. We describe and analyze the pair $(A_{\mathrm{comp}}, P_{\mathrm{dec}})$, which uses the commit-and-evaluate protocol $(P_{\mathrm{commit}}, P_{\mathrm{eval}})$ of Lemma 2.23 with fixed input p and resilience parameter

$s = s(m,n) = (100q)^{-1}$, where $q = q(m,n) = \mathsf{polylog}(T(n),m)$ 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 2.23. 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 bit b and a message for P_{commit}, which produces a commitment π_{y_i}. 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 $|S^2| = \log(1/s)\cdot\mathsf{poly}(\Delta,r) = \mathsf{poly}(m,\log T(n))$ points in $\mathbb{F}^{r'}$ together with evaluations of $\hat{z}$ on each point, each of which can be described with $\log p = \mathsf{polylog}(m,\log T(n))$ bits, resulting in a total output length of $\mathsf{poly}(m,\log T(n))$.

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 2.23. 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 2.23 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, the commit phase is successful for $\hat{z}$ and $\tilde{y}_i$ with probability at least $1 - 2s \geq 0.99$ for sufficiently large m and n. 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.

Assuming the first two "bad" events do not happen, the only way Merlin could try to have Arthur output a wrong value 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 4.11 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 $\log(1/s) \cdot \mathsf{poly}(\Delta, r) = \mathsf{poly}(m, \log T(n))$ evaluations of $\hat{z}$. Computing $\tilde{z}$ takes time $n \cdot \mathsf{polylog}(n) = n \cdot \mathsf{polylog}(T(n))$, and computing each evaluation of $\hat{z}$ takes time $n \cdot \mathsf{poly}(h, \log p, r, \log n) = n \cdot \mathsf{poly}(m, \log T(n))$, resulting in a total running time of $n \cdot \mathsf{poly}(m, \log T(n))$.

As for P_{dec}, the commit phase takes time $\log(1/s) \cdot \mathsf{poly}(\Delta, r) = \mathsf{poly}(m, \log T(n))$ and

two rounds of communication. Afterwards, evaluating each query made by V with P_{eval} takes time $(\log(1/s))^2 \cdot \Delta^{O((\log r)^2)} = (m \cdot \mathsf{polylog}(T(n)))^{O((\log r)^2)}$. The verification step for V takes time $\mathsf{poly}(m, \log T(n))$, and it makes at most $q = \mathsf{polylog}(m, T(n))$ 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 two rounds 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 $\mathsf{polylog}(T(n))$ 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 2.23 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. ■

If the function f in Theorem 4.13 is computable in deterministic time T, then we can modify the construction so that the generator H runs in *deterministic* time $\mathsf{poly}(T(n), m)$. To do so, we similarly define L_f as the set of $(\tilde{z}, n, i, b)$ such that $\text{Dec}(\tilde{z}) = z$ for some $z \in \{0,1\}^n$ and $f(z)_i = b$, and employ the canonical PCPP construction of Lemma 4.12, interpreting (n, i, b) as the explicit input and $\tilde{z}$ as the implicit input. With these changes, it follows that $L_f \in \mathsf{DTIME}[n \cdot \mathsf{polylog}(n) + T(n)]$, 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. We thus establish Corollary 4.14.

Corollary 4.14. *Let $T(n)$ be a time bound and f a function computable in deterministic time $T(n)$. There exists a deterministic algorithm H and a pair P_{rec} consisting of a probabilistic algorithm A_{comp} and a promise Arthur-Merlin protocol P_{dec} that have the same properties as in Theorem 4.13, with the only difference being that H is deterministic instead of nondeterministic.*

4.4.2 From refutation to derandomization

We now show that the third item in Theorem 4.2 implies the first 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 4.13 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 4.13 and output the union of the sets obtained. Provided that n is a sufficiently large polynomial in m, the reconstructor meets the resource and compression bounds for a $\mathsf{prAMTICOMP}[n^{a+\epsilon}, n^\epsilon]$ 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 4.13 fails for z_i). It follows that $H(z_i, D)$ hits D (item 1 in Theorem 4.13 holds).

Theorem 4.15. *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 $\mathsf{prAMTICOMP}[n^{a+\epsilon}, n^\epsilon]$ 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 $\mathsf{poly}(T(\mathsf{poly}(n)))$.*

Proof. Let $(A_{\text{comp}}, P_{\text{dec}})$ be the reconstructor of Theorem 4.13 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 $A_{\text{comp}}(\cdot, 1^m)$ denote algorithm A_{comp} with 1^m fixed as its second input and similarly let $P_{\text{dec}}(\cdot, D)$ be the protocol P_{dec} with the circuit D fixed as its second input. The generator then feeds inputs 1^n and $(A_{\text{comp}}(\cdot, 1^m), P_{\text{dec}}(\cdot, D))$ into the refuter R to obtain a list of inputs $(z_1, \ldots, z_\tau)$. Finally, the generator outputs $\cup_{i \in [\tau]} H(z_i, D)$, where H is the generator of Theorem 4.13 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 4.13 guarantees that $(A_{\text{comp}}(\cdot, 1^m), P_{\text{dec}}(\cdot, D))$ is sound for f. To ensure correctness of the generator, we set the value of n sufficiently large such that $A_{\text{comp}}(\cdot, 1^m)$ and $P_{\text{dec}}(\cdot, D)$ run in time at most $n^{a+\epsilon}$ and such that the output length of $A_{\text{comp}}(\cdot, 1^m)$ is at most n^ϵ. In this case, the refuter must output, on every accepting computation path, a list of strings $(z_1, \ldots, z_\tau)$ that contains at least one z_i such that $(A_{\text{comp}}(\cdot, 1^m), P_{\text{dec}}(\cdot, D))$ fails to compute $f(z_i)$ with completeness 1 and soundness 1/3. This means that item 2 in Theorem 4.13 fails for $z = z_i$, and therefore item 1 must hold, which implies that our targeted generator hits D.

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

By setting $k \geq 2k_1/\epsilon$, it holds for sufficiently large m and any input of length $\ell \geq n = m^k$ that the string output by $A_{\text{comp}}(\cdot, 1^m)$ has length at most ℓ^ϵ. Similarly, setting $k \geq 2k_2/\epsilon$, it holds for sufficiently large m and any input of length $\ell \geq n = m^k$ that the running time

of $A_{\mathrm{comp}}(\cdot, 1^m)$ is at most $\ell^{1+\epsilon} \leq \ell^{a+\epsilon}$. Finally, setting $k \geq 2k_3 \cdot (\log(ak))^2/\epsilon$, which holds for sufficiently large constant k, guarantees that $P_{\mathrm{dec}}(\cdot, D)$ runs in time at most $\ell^{a+\epsilon}$ for $\ell \geq n = m^k$ 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 $T(n + m') = T(\mathsf{poly}(m))$, 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 $\mathsf{poly}(T(\mathsf{poly}(m)))$, which dominates the running time for the generator. ∎

In contrast to the corresponding result of [CTW23] in the BPP setting, Theorem 4.15 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 hitting-set generator 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 $2^{\mathsf{polylog}(n)}$ and compression lengths of the form $2^{(\log n)^\epsilon}$ for $\epsilon \in (0, 1)$.

Theorem 4.16. *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 $\mathsf{prAMTICOMP}[2^{(\log n)^{a+\epsilon}}, 2^{(\log n)^\epsilon}]$ with promised soundness for f such that R runs*

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

4.4.3 From derandomization to refutation

Next, we prove that the first item in Theorem 4.2 implies the second one. In fact, we establish something stronger: Assuming the existence of a targeted hitting-set generator as in the first 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 4.17). The second item then follows as the identity function has such a probabilistic refuter (Proposition 4.18).

A probabilistic refuter is a refuter that produces a counterexample with constant probability over its internal randomness. In the case of the class $\mathsf{prAMTICOMP}[t(n), \gamma(n)]$ 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 $\gamma(\ell)$, 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 4.17). 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 4.3.2. 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 4.13 has perfect completeness. The resilient soundness property of the reconstructor in Theorem 4.13 ensures that we only need to worry about refuting pairs $(A_{\text{comp}}, P_{\text{dec}})$ that are sound for f.

Theorem 4.17. *Assume there exists 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$[t(n), \gamma(n)]$

protocols with promised soundness for f. Then there exists a list-refuter R for f against $\mathsf{prAMTICOMP}[t(n), \gamma(n)]$ *protocols with promised soundness for f such that R is computable in nondeterministic time $T(\mathsf{poly}(m, t(\mathsf{poly}(n))))$, 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 4.17 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 $\mathsf{poly}(m, n)$.

Proof of Theorem 4.17. Let H be the hypothesized targeted hitting-set generator. 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, H 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 $\mathsf{prAMTICOMP}[t(n), \gamma(n)]$ 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 comprises of 1^n and a pair $(A_{\mathrm{comp}}, P_{\mathrm{dec}})$ 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 $R_{\mathrm{pr}}(1^n, (A_{\mathrm{comp}}, P_{\mathrm{dec}}); r_1)$ to obtain an input z of length ℓ with $n \leq \ell = O(\mathsf{poly}(n))$. 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 $A'_{\mathrm{comp}}(z, r_2)$, co-nondeterministically verifies that there is no Merlin message that would lead P'_{dec} with input $A'_{\mathrm{comp}}(z, r_2)$ 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 $(A_{\mathrm{comp}}, P_{\mathrm{dec}})$ 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, $R_{\mathrm{pr}}(1^n, (A_{\mathrm{comp}}, P_{\mathrm{dec}}); r_1)$ outputs, with probability at least δ over a random choice of r_1, an input z of length $\ell \geq n$ such that $(A_{\mathrm{comp}}, P_{\mathrm{dec}})$ 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 $P_{\mathrm{dec}}(A_{\mathrm{comp}}(z, r_2); r_3)$ 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 $R_{\mathrm{pr}}(1^n, A_{\mathrm{comp}}, P_{\mathrm{dec}}; r_1)$ 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 $(A_{\mathrm{comp}}, P_{\mathrm{dec}})$ 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 $z = R_{\mathrm{pr}}(1^n, A_{\mathrm{comp}}, P_{\mathrm{dec}}; r_1)$. As such a z is on the list output by R on every accepting computation path, R is a list-refuter for f against $\mathsf{prAMTICOMP}[t(n), \gamma(n)]$ protocols with promised soundness for f.

Let m denote the description length of $(A_{\mathrm{comp}}, P_{\mathrm{dec}})$. The co-nondeterministic circuit D' constructed by R on inputs 1^n and $(A_{\mathrm{comp}}, P_{\mathrm{dec}})$ has size $\mathsf{poly}(m, n, t(\mathsf{poly}(n))) = \mathsf{poly}(m, t(\mathsf{poly}(n)))$ since $t(n) \geq n$, and thus the time required for computing $H(D')$ is $T(\mathsf{poly}(m, t(\mathsf{poly}(n))))$. Finally, R needs to compute $R_{\mathrm{pr}}(1^n, A_{\mathrm{comp}}, P_{\mathrm{dec}}; r_1)$ for at most $T(\mathsf{poly}(m, t(\mathsf{poly}(n))))$ strings r_1, and each such execution takes time $\mathsf{poly}(m, n)$. The final running time is thus $T(\mathsf{poly}(m, t(\mathsf{poly}(n)))) + \mathsf{poly}(m, n) = T(\mathsf{poly}(m, t(\mathsf{poly}(n))))$ 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 4.18. *For every constant $\epsilon \in (0,1)$, there exists a probabilistic polynomial-time refuter for the identity function against* $\mathsf{prAMTICOMP}[\infty, n^\epsilon]$ *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. ∎

4.4.4 Proof of equivalence result

We now have all the steps involved in Theorem 4.2; we just need to tie them together.

Proof of Theorem 4.2. The implication 2 $\implies$ 3 holds trivially by taking the identity for f. The implication 3 $\implies$ 1 is Theorem 4.15. The implication 1 $\implies$ 2 follows by combining Theorem 4.17 and Proposition 4.18 with polynomial time bounds. ∎

When the function f is computable deterministically, the strategy also applies to other types of targeted hitting-set generators such as deterministic, single-valued and deterministic with parallel NP oracle. We present a general version of Theorem 4.2 that applies to different types of targeted hitting-set generators for prAM, and is useful for obtaining our explicit construction results.

Theorem 4.19. *Let* $\mathcal{C} \in \{\mathsf{P}, (\mathsf{NP} \cap \mathsf{coNP}), \mathsf{P}_{\|}^{\mathsf{NP}}\}$. *The following are equivalent:*

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

Proof (sketch). The equivalence goes along the same lines as that of Theorem 4.2, using Corollary 4.14 following the observation after Theorem 4.13 that if the algorithm computing f is deterministic, then the targeted hitting-set generator construction of Theorem 4.13 is also deterministic. In this case, assuming a refuter computable in $\mathcal{C}$ and following Theorem 4.15, we obtain a targeted hitting-set generator that is also computable in $\mathcal{C}$. Similarly and following Theorem 4.17, a targeted hitting-set generator computable in $\mathcal{C}$ leads to a refuter computable in $\mathcal{C}$. ∎

The equivalence of Theorem 4.19 scales in the same way as that of Theorem 4.2, and in particular holds for quasipolynomial and subexponential time bounds.

4.5 Mild derandomization

In this section, we prove the more general equivalence of Theorem 4.3 in the setting of mild derandomization.

4.5.1 From refutation to derandomization

We start by proving a generic version for the implication of refutation to derandomization. We show that derandomization follows from the existence of the same type of (weaker) refuter considered for Theorem 4.2: List-refuters that are only guaranteed to produce a counterexample for sound bottleneck protocols.

Theorem 4.20. *Let T be a time bound, $\mathcal{C}$ a machine class in $\{\mathsf{P}^{\mathsf{NP}}, \mathsf{ZPP}^{\mathsf{NP}}, \Sigma_2\mathsf{P}\}$, a a constant and f a function computable in nondeterministic time n^a. If for some constant $\epsilon \in (0,1)$ there is a list-refuter R for f against $\mathsf{prAMTICOMP}[n^{a+\epsilon}, n^{\epsilon}]$ with promised soundness for f such that R is computable in time T by machines of type $\mathcal{C}$, then there is a targeted hitting-set generator for prAM that is computable in time $\mathsf{poly}(T(\mathsf{poly}(m)))$ by machines of type $\mathcal{C}$.*

Proof (sketch). The argument for Theorem 4.15 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 $\mathsf{poly}(T(\mathsf{poly}(n)))$. To obtain the targeted hitting-set generator of Theorem 4.15, we run the refuter to obtain a list of inputs of length $n = \mathsf{poly}(m)$ (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 construction of Theorem 4.13, which runs in nondeterministic time polynomial in $T(n)$. For a refuter computable by a Σ_2 algorithm, we can have the targeted generator guess the output of the refuter and of the generator construction, and then verify with a nondeterministic and universal guess that the outputs of the refuter and the generator construction were guessed correctly. For deterministic and zero-error probabilistic algorithms with oracle access to SAT, we can use the SAT oracle to compute, after obtaining the output of the refuter, the lexicographically least output of

the nondeterministic generator construction. In either case, the final running time for the targeted HSG is $\mathsf{poly}(T(\mathsf{poly}(m)))$. ■

4.5.2 From derandomization to refutation

We show that a more general version of the implication of derandomization to refutation of Theorem 4.3 holds. For this version, we now consider stronger refuters than those of Theorem 4.2: Refuters that output a single counterexample and that are guaranteed to work against every bottleneck protocol. This implication essentially follows from the equivalence between leakage-resilient hardness and the existence of refuters against the identity function, together with the results in Chapter 3. Formally establishing the equivalence, however, involves some technical details that we believe detracts from the main idea, and thus we present a direct proof using the techniques in Chapter 3.

Theorem 4.21. *Let T be a time bound and $\mathcal{C}$ denote a machine class in $\{\mathsf{P}^{\mathsf{NP}}, \mathsf{ZPP}^{\mathsf{NP}}, \Sigma_2\mathsf{P}\}$. If $\mathsf{prAMTIME}[n]$ can be simulated by algorithms of type $\mathcal{C}$ in time T, then for any constant $\epsilon \in (0,1)$ there exists a refuter computable in time $\mathsf{poly}(m,n) \cdot T(\mathsf{poly}(m,n))$ for the identity function against $\mathsf{prAMTICOMP}[n^{1+\epsilon}, n^{\epsilon}]$.*

Proof. Following the strategy of Section 3.5.2, we describe a $\mathsf{prBPP}_{||}^{\mathsf{SAT}}$-search problem that captures the refutation task, and then derandomize it using the derandomization assumption. Fix $\epsilon \in (0,1)$ and let 1^n and $(A_{\text{comp}}, P_{\text{dec}})$ of description length m be inputs for the refuter. We define the following search problem (R_Y, R_N):

- $(1^n, A_{\text{comp}}, P_{\text{dec}}, z) \in R_Y$ if and only if $|z| \geq n$ and $(A'_{\text{comp}}, P'_{\text{dec}})$ fails to output z with completeness $3/4$ and soundness $1/4$.
- $(1^n, A_{\text{comp}}, P_{\text{dec}}, z) \in R_N$ if and only if $|z| \geq n$ and $(A'_{\text{comp}}, P'_{\text{dec}})$ outputs z with completeness 1 and soundness $1/3$.

Where A'_{comp} and P'_{dec} denote, respectively, A_{comp} and P_{dec} clocked to run in time $\ell^{1+\epsilon}$ and with enforced compression length ℓ^{ϵ} on inputs of length ℓ. Note that any z such that $(1^n, A_{\text{comp}}, P_{\text{dec}}, z) \notin R_N$ serves as counterexample for $(A_{\text{comp}}, P_{\text{dec}})$.

To compute a solution, following Proposition 4.18, it suffices to output a random string z of length $\ell = \Theta(n)$.

To distinguish between yes and no instances, a probabilistic algorithm with non-adaptive oracle access to SAT can estimate the completeness and soundness by asking the following queries for constantly many random pairs of random strings (r_1, r_2):

1. Is there a Merlin response that would lead $P'_{\text{dec}}(A'_{\text{comp}}(z; r_1); r_2)$ to succeed and output z?

2. Is there a Merlin response that would lead $P'_{\text{dec}}(A'_{\text{comp}}(z; r_1); r_2)$ to succeed and output a string different than z?

With a constant number of queries of each type, it is possible to attain high constant probability of success and small constant error, say, $\delta = 0.01$. Let $\tilde{c}$, $\tilde{s}$ be the completeness and soundness estimates, respectively. If $\tilde{c} \leq 0.9$ or $\tilde{s} \geq 0.1$, then the verification algorithm accepts, otherwise it rejects. The running time for the verification algorithm is polynomial in n and the description length m for the given bottleneck protocol.

We now argue the result for deterministic and zero-error probabilistic algorithms with oracle access to SAT. By Proposition 3.21, there exists a polynomial-time (in m and n) deterministic algorithm with oracle access to a problem in $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$ that, on input $(1^n, A_{\text{comp}}, P_{\text{dec}})$, outputs a counterexample z of length greater than or equal to n. Since $\mathsf{prBPP}_{\|}^{\mathsf{SAT}} \subseteq \mathsf{P}_{\|}^{\mathsf{prAM}}$ and due to the derandomization assumption for prAM, the oracle can be simulated in $\mathsf{DTIME}[T]^{\mathsf{SAT}}$ (or $\mathsf{ZPTIME}[T]^{\mathsf{SAT}}$), and thus the entire procedure can be executed in $\mathsf{DTIME}[\mathsf{poly}(m,n) \cdot T(\mathsf{poly}(m,n))]^{\mathsf{SAT}}$ (or $\mathsf{ZPTIME}[\mathsf{poly}(m,n) \cdot T(\mathsf{poly}(m,n))]^{\mathsf{SAT}}$).

For the Σ_2 machine model, the part of the proof that shows (R_Y, R_N) is a $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$-search problem shows, in particular, that for all $(1^n, A_{\text{comp}}, P_{\text{dec}})$ such that $(A_{\text{comp}}, P_{\text{dec}})$

respect the time and compression upper bounds, there exists a "good" counterexample z, and that verifying whether a z is "good" can be done in $\mathsf{prBPP}_{\|}^{\mathsf{SAT}}$. Thus, under the derandomization assumption and using Proposition 3.24, a Σ_2 algorithm can guess a candidate z and then verify that it is a "good" counterexample set in time $\mathsf{poly}(m,n)\cdot T(\mathsf{poly}(m,n))$. ■

As we now have all the steps involved in Theorem 4.3, we provide a proof for it.

Proof of Theorem 4.3. We established the equivalence $1 \iff 2$ in Theorem 4.3. That $1 \implies 3$ follows from Theorem 4.21 with polynomial time bounds. That $3 \implies 4$ is trivial, and the implication $4 \implies 1$ follows from Theorem 4.20 with polynomial time bounds. ■

4.5.3 Extreme compression and circuit lower bounds

To formally state Theorem 4.4, we need the notion of a low-end io-refuter. To ease into the definition of such a refuter and the proof of Theorem 4.4, we first state and prove a simpler version of the theorem. In the simpler version, we assume the existence of refuters that work almost-everywhere, and conclude derandomization that works almost-everywhere. In contrast, the refuters and derandomization of Theorem 4.4 are only guaranteed to work for infinitely-many inputs. For a time bound $t(n)$, we define the $t(n)$-local bottleneck algorithms as pairs $(A_{\text{comp}}, A_{\text{dec}})$ where A_{dec} computes each bit of its output in time $t(n)$. To capture the notion of a refuter against $\mathsf{polylog}(n)$ compression length and derandomization in subexponential time 2^{n^ϵ} for all $\epsilon > 0$, we allow for multiple refuters, one for each ϵ and constant k capturing compression length and local time $(\log n)^k$. Recall that a refuter receives as input 1^n and an algorithm of description length m, and is supposed to output a counterexample of length at least n.

Theorem 4.22. *Assume there exists a constant c such that for all $k > 0$, there exists a refuter R_k for the identity function against $(\log n)^k$-local $\mathsf{prBPTICOMP}[n\cdot(\log n)^k, (\log n)^k]_{\|}^{\mathsf{SAT}}$, and that each R_k is computable in $\Sigma_2\mathsf{TIME}[(m\cdot n)^c]$. Then, for all $\epsilon > 0$, $\mathsf{prAM} \subseteq \Sigma_2\mathsf{TIME}[2^{n^\epsilon}]$.*

Proof. We show, under the assumption, that for all $\epsilon > 0$ there exists a targeted hitting-set generator for prAM that runs in Σ_2-time $2^{O(m^\epsilon)}$, and the derandomization result follows.

Fix $\epsilon > 0$ and let $k = \Theta(1/\epsilon)$ be a constant to be determined later. Let $(A_{\text{comp}}, A_{\text{dec}})$ be the reconstructor of Lemma 3.10. We first describe the operation of the targeted hitting-set generator, then we analyze its correctness and running time.

Generator. Let $A_{\text{comp}}(\cdot, 1^m)$ denote algorithm A_{comp} with 1^m fixed as its second input and similarly let $A_{\text{dec}}(\cdot, D)$ be the algorithm A_{dec} with the circuit D fixed as its second input. The generator, on input a co-nondeterministic circuit D of size m, first sets $n = 2^{m^\epsilon}$. The generator then feeds inputs 1^n and $(A_{\text{comp}}(\cdot, 1^m), P_{\text{dec}}(\cdot, D))$ into the refuter $R_{\epsilon,k}$ to obtain $z \in \{0,1\}^\ell$ for $\ell \geq n$. Finally, the generator outputs $H_{\text{det}}(z, 1^m)$, where H_{det} is the generator of Lemma 3.10.

Correctness. To ensure correctness of the generator, we set the value of k such that, on inputs of length $\ell \geq n$, $A_{\text{comp}}(\cdot, 1^m)$ and $A_{\text{dec}}(\cdot, D)$ run in time at most $\ell \cdot (\log \ell)^k$ and such that the output length of $A_{\text{comp}}(\cdot, 1^m)$ is at most $(\log \ell)^k$. In this case, the refuter must output a string z such that $(A_{\text{comp}}(\cdot, 1^m), A_{\text{dec}}(\cdot, D))$ fails to compute identity on z. This means that item 2 in Lemma 3.10 fails for z, and therefore item 1 must hold, and thus our targeted generator hits D.

We now set the value of k. On input $z \in \{0,1\}^\ell$, $A_{\text{comp}}(\cdot, 1^m)$ outputs a string of length $\mathsf{poly}(m, \log \ell)$. Moreover, the running time for $A_{\text{comp}}(\cdot, 1^m)$ is $\ell \cdot \mathsf{poly}(m, \log \ell)$, and the running time for $A_{\text{dec}}(\cdot, D)$ is $\ell \cdot \mathsf{poly}(m, \log \ell)$. Let k' be a constant such that $A_{\text{comp}}(\cdot, 1^m)$ outputs a string of length at most $(m \cdot \log \ell)^{k'}$, and such that the running time for both $A_{\text{comp}}(\cdot, 1^m)$ and $A_{\text{dec}}(\cdot, D)$ is at most $\ell \cdot (m \cdot \log \ell)^{k'}$. By setting $k = 2k'/\epsilon$, it holds for sufficiently large m and any input of length $\ell \geq n = 2^{m^\epsilon}$ that the string output by $A_{\text{comp}}(\cdot, 1^m)$ has length at most $(\log \ell)^k$. Similarly, it holds for sufficiently large m and any input of length $\ell \geq n = 2^{m^\epsilon}$ that the running time of $A_{\text{comp}}(\cdot, 1^m)$ and $A_{\text{dec}}(\cdot, D)$ is at most $\ell \cdot (\log \ell)^k$.

Running time. Let $d(m) = \Theta(m \cdot \log m)$ denote the description length of $(A_{\text{comp}}, A_{\text{dec}})$. Computing the output z of the refuter R_k takes Σ_2-time $n^c \cdot 2^{c \cdot d(m)^\epsilon} = 2^{m^{O(\epsilon)}}$, which also

serves as an upper bound for the length of z. Then, computing $H(z, 1^m)$ takes deterministic time $\mathsf{poly}(2^{m^{O(\epsilon)}}, m) = 2^{m^{O(\epsilon)}}$, which dominates the running time for the generator. ■

We observe that the refuter assumed in Theorem 4.22 is overkill for the derandomization task. Notice that, in the proof of Theorem 4.22, the targeted hitting-set generator sets $n = 2^{m^\epsilon}$ when running the refuter. In the low end and using Lemma 3.10, this is necessary since the compression length for the reconstructor of Lemma 3.10 is polynomial in m, and thus we cannot hope to obtain polylogarithmic compression length with n closer to m. In that case, it makes sense to consider refuters that are only guaranteed to work when n is sufficiently large in relation to m, which is what we do with low-end refuters.

In the infinitely-often setting of Theorem 4.4, we would like the targeted hitting-set generator to work for infinitely-many circuit sizes m. Following the same strategy as the proof of Theorem 4.22, this requires a refuter with the following behavior: For infinitely m, the refuter outputs a valid counterexample on input a bottleneck algorithm of description length m and 1^n for n around 2^{m^ϵ}. Motivated by this application to low-end derandomization, we define low-end io-refuters.

Definition 4.23. Let $f : \{0,1\}^* \to \{0,1\}^*$ be a total function, $\mathcal{A}$ a resource-bounded semantic class of algorithms and ϵ a constant. A low-end io-refuter R with parameter ϵ 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 string z of length at least n. For infinitely many description lengths m, the following holds: There exists $n \in [2^{m^\epsilon}, 2^{(m+1)^\epsilon}]$ such that, if A satisfies the resource bounds of $\mathcal{A}$ for all inputs of length at least n, then $\mathcal{A}$ fails to compute $f(z)$. ◂

Since we only care about the case $n \geq 2^{m^\epsilon}$, we measure the running time for the refuters in terms of n only, assuming implicitly that $n \geq 2^{m^\epsilon}$.

We are now able to formally state Theorem 4.4.

Theorem 4.24 (Formal version of Theorem 4.4). *The following are equivalent.*

1. *For all $\epsilon > 0$,* $\mathsf{prAM} \subseteq \text{io-}\Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$.

2. $\Sigma_2\mathsf{EXP} \not\subseteq \mathsf{NP/poly}$.

3. *There exists a constant c such that for all $\epsilon, k > 0$, there exists a low-end io-refuter $R_{k,\epsilon}$ with parameter ϵ for the identity function against $(\log n)^k$-local $\mathsf{prBPTICOMP}[n \cdot (\log n)^k, (\log n)^k]^{\mathsf{SAT}}_{\|}$. Each $R_{k,\epsilon}$ is computable in Σ_2-time n^c with $\log n$ bits of advice.*

We prove that2 $\implies$ 3 in Lemma 4.25, and that 3 $\implies$ 1 in Lemma 4.26. That 1 and 2 are equivalent follows from [AvM17].

Given the equivalence between leakage-resilient hardness and refuters for the identity function, the intuition for the proof of Theorem 4.24 is very similar to that for Theorem 3.7. As a first step, we show the lower bound $\Sigma_2\mathsf{E} \not\subseteq \mathsf{NP/poly}$ implies the existence of refuters for the identity function against non-adaptive SAT-oracle bottleneck algorithms. First, by Lemma 3.35, the non-uniform lower bound above implies that $\Sigma_2\mathsf{E} \not\subseteq \mathsf{P}^{\mathsf{SAT}}_{\|}\mathsf{/poly}$. The refuter, on input 1^n and any bottleneck algorithm $(A_{\mathrm{comp}}, A_{\mathrm{dec}})$, outputs the truth-table z of a language $L \in \Sigma_2\mathsf{E}$ with high circuit complexity at input length $\log n$. With $\log n$ bits of advice (indicating how many strings of length $\log n$ are in L), the refuter can be computed in Σ_2-time $2^{O(\log n)} = \mathsf{poly}(n)$. Assume that, for n around 2^{m^ϵ}, the compression and time bounds hold for $(A_{\mathrm{comp}}, A_{\mathrm{dec}})$ on inputs of length at least n, yet z is not a good counterexample for $(A_{\mathrm{comp}}, A_{\mathrm{dec}})$. By fixing a "good" output π_z of $A_{\mathrm{comp}}(z)$ into A_{dec}, as well as a "good" random string for A_{dec} and following Adleman's argument [Adl78], we get a polynomial-size probabilistic non-adaptive SAT-oracle circuit of size $\mathsf{poly}(m)$ that computes L on input length $\log n$. Given the assumed circuit lower bound, it follows that the refuter must output valid counterexamples for infinitely-many m (and n around 2^{m^ϵ}). We now formalize this argument.

Lemma 4.25. *Assume $\Sigma_2\mathsf{E} \not\subseteq \mathsf{NP/poly}$. Then for all $\epsilon, k > 0$, there exists a low-end io-refuter $R_{k,\epsilon}$ with parameter ϵ for the identity function against the class of $(\log n)^k$-local $\mathsf{prBPTICOMP}[\infty, (\log n)^k]^{\mathsf{SAT}}_{\|}$ algorithms. Each $R_{k,\epsilon}$ is computable in Σ_2-time n^c with $\log n$ bits of advice, for a fixed constant c.*

Proof. Let $L \in \Sigma_2\mathsf{E} \setminus \mathsf{NP/poly}$ and fix $\epsilon, k > 0$. On input $(A_{\text{comp}}, A_{\text{dec}})$ of description length m and 1^n (with $n \geq 2^{m^\epsilon}$), the refuter $R_{\epsilon,k}$ outputs the truth-table of L at input length $\log n$.

First, notice that, assuming $n \geq 2^{m^\epsilon}$, $R_{\epsilon,k}$ can be computed with $\log n$ bits of advice, indicating how many strings of length n are in L, in Σ_2-time n^c, where c is the constant such that $L \in \Sigma_2\mathsf{TIME}[2^{cn}]$.

Now, assume that for almost-all m and all $n \in [2^{m^\epsilon}, 2^{(m+1)^\epsilon}]$, the string z output by $R_{\epsilon,k}$ on input 1^n and $(A_{\text{comp}}, A_{\text{dec}})$ of description length m fails as a counterexample, i.e., the compression and time bounds hold for every string of length $\ell \geq n$, yet it holds that $(A_{\text{comp}}, A_{\text{dec}})$ computes the identity function on input z. By an averaging argument, for all $n \in [2^{m^\epsilon}, 2^{(m+1)^\epsilon}]$, there exists an output π_z of $A_{\text{comp}}(z)$ of length at most $(\log n)^k$ such that $A_{\text{dec}}(\pi_z)$ computes individual bits of z, i.e., decides the language L on input length $\log n$, with high probability in time $(\log n)^k$. Since $n \geq 2^{m^\epsilon}$ and $A_{\text{dec}}(\pi_z)$ can be transformed into a circuit of size $\mathsf{poly}(m, (\log n)^k)$, we get that $L \in \mathsf{BPP}_{||}^{\mathsf{SAT}}\mathsf{/poly}$, and thus $L \in \mathsf{P}_{||}^{\mathsf{SAT}}\mathsf{/poly}$ [Adl78]. By Lemma 3.35, $L \in \mathsf{NP/poly}$. As this is a contradiction, it follows that for infinitely many m there must exist $n \in [2^{m^\epsilon}, 2^{(m+1)^\epsilon}]$ such that the counterexample z output by $R_{\epsilon,k}$ is valid. ■

Now, we show that a slightly weaker hardness assumption, where the refuter only needs to provide counterexamples against $(\log n)^k$-local $\mathsf{prBPTICOMP}[n\cdot(\log n)^k, (\log n)^k]_{||}^{\mathsf{SAT}}$, suffices to obtain low-end derandomization of prAM.

Lemma 4.26. *Assume there exists a constant c such that for all $\epsilon, k > 0$, there exists a low-end io-refuter $R_{k,\epsilon}$ with parameter ϵ for the identity function against $(\log n)^k$-local $\mathsf{prBPTICOMP}[n \cdot (\log n)^k, (\log n)^k]_{||}^{\mathsf{SAT}}$, and that each $R_{k,\epsilon}$ is computable in Σ_2-time n^c with $\log n$ bits of advice. Then, $\mathsf{prAM} \subseteq \text{io-}\Sigma_2\mathsf{TIME}[2^{n^\epsilon}]/n^\epsilon$.*

Proof. The proof follows the proof of Theorem 4.22 closely, with just a few differences. The main difference is that now, after fixing $\epsilon > 0$, our objective is to obtain a targeted hitting-set generator H for prAM that runs in $\Sigma_2\mathsf{TIME}[2^{O(m^\epsilon)}]/m^{O(\epsilon)}$.

On input a co-nondeterministic circuit D of size m, let $d(m) = \Theta(m \cdot \log m)$ denote the description length of $(A_{\text{comp}}(\cdot, 1^m), A_{\text{dec}}(\cdot, D))$. Let also k be the same constant as in the proof of Theorem 4.22. If there exists $m' \in [d(m), d(m+1)]$ and $n \in [2^{(m')^\epsilon}, 2^{(m'+1)^\epsilon}]$ such that the refuter $R_{\epsilon,k}$ outputs a valid counterexample on input 1^n and any bottleneck protocol of description length m' (that is, m' is one of the infinitely-many "good" input lengths for the refuter), then H pads $(A_{\text{comp}}(\cdot, 1^m), A_{\text{dec}}(\cdot, D))$ to length exactly m' and operates exactly as in Theorem 4.22: It computes $z = R_{\epsilon,k}(1^n, (A_{\text{comp}}(\cdot, 1^m), A_{\text{dec}}(\cdot, D)))$ and uses z as the input for the generator of Lemma 3.10. Otherwise, H outputs $\{0^n\}$.

Because the intervals $[d(m), d(m+1)]$ for all m cover all but finitely-many input lengths, and $R_{\epsilon,k}$ is guaranteed to output valid counterexamples for some n in the considered range for infinitely many m', H is guaranteed to work for infinitely many circuit sizes m. Identifying m' and n requires $O(\log m + \log n) = m^{O(\epsilon)}$ bits of advice, and the running time for H is $2^{O(m^\epsilon)}$, which concludes the proof. ∎

4.6 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 4.6.1 and then highlight some applications in Section 4.6.2.

4.6.1 General statement

We prove a more general version of Proposition 4.6 that applies to 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 4.27. *Let T be a time bound and $\mathcal{C} \in \{\mathsf{P}, (\mathsf{NP} \cap \mathsf{coNP}), \mathsf{NP}, \mathsf{P}_{\|}^{\mathsf{NP}}\}$. 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* $\mathsf{poly}(T(\mathsf{poly}(n)))$ *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 $\Pi \in \mathsf{coNP}$, 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 $m = \mathsf{poly}(n)$, and it can be constructed in that time. Then, computing $H(D)$ takes time $T(\mathsf{poly}(n))$, and thus produces a list of strings of size at most $T(\mathsf{poly}(n))$. Finally, running B on the resulting list takes time $\mathsf{poly}(T(\mathsf{poly}(n)))$.

For the case $\mathcal{C} = \mathsf{P}_{\|}^{\mathsf{NP}}$, the construction above has two rounds of non-adaptive NP queries. Using Fact 3.22, we can obtain a construction with a single round of non-adaptive NP queries while incurring a polynomial time overhead. ∎

4.6.2 Instantiations

In this section, we provide examples of explicit constructions that can be obtained by applying Proposition 4.27.

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 4.5.

Theorem 4.28. *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* $T(\mathsf{poly}(m))$ *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 4.27. 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 greater than or equal to s. ∎

Theorem 4.5 follows from Theorem 4.28 and traditional hardness vs. randomness tradeoffs [Uma03]. The conclusion of Theorem 4.28 together with the construction in [Uma03]

results in a nondeterministic PRG that, on input 1^m, runs in time $\mathsf{poly}(T(\mathsf{poly}(m))$ 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 4.29. *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 $\mathsf{prAMTICOMP}[n^{a+\epsilon}, n^\epsilon]$ 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 4.19 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 4.27. 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. ∎

The reason we present the precondition of Theorem 4.29 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; GST03].

4.7 Further research

In this chapter, we fully characterized derandomization of prAM via targeted generators in terms of a refutation task against protocols that go through a compression phase. With the goal of characterizing any whitebox derandomization for prAM, the most important problem that is left open is obtaining an equivalence between derandomization and targeted generators for prAM. As mentioned in the introduction, the techniques employed to obtain such an equivalence for prBPP [Gol11], which are also used in all equivalences involving derandomization for prBPP [LP22; LP23; Kor22a; CTW23], do not seem compatible with the prAM setting. The reason for this is that a major component for these arguments is employing a derandomized algorithm as an oracle, which is fine for prBPP since P = coP, but does not work for prAM since it is unknown whether NP = coNP.

We do remark, however, that we were able to avoid the complementation issue above in some settings. In Chapter 3, we did so by relaxing the derandomization and uniformizing a classical pseudorandom generator construction. In this chapter, we were able to do so by relying on the resilient soundness property of the reconstructor for our targeted hitting-set generator construction.

In the next chapter, we obtain further applications of the refutation and compression framework in the space-bounded setting.

Chapter 5

Space-Bounded Computation

5.1 Introduction

In this chapter, we apply the techniques we have developed in the previous chapters to space-bounded computation in the settings of isolation and catalytic algorithms.

Isolation involves pinpointing a solution to a problem, out of the many that it could have. In a computational context, isolation refers to efficient simulations by unambiguous algorithms: nondeterministic algorithms that either have a unique accepting computational path or none. For a resource-bounded class $\mathcal{C}$, we typically aim for isolations where the unambiguous algorithm is also in $\mathcal{C}$. In this chapter we focus on isolation for two classes:

- The class NL of logspace nondeterministic algorithms.
- The class CNL of the so-called logspace nondeterministic *catalytic* algorithms.

Space-bounded isolation. In the space-bounded setting, the main open problem is whether $\mathsf{NL} = \mathsf{UL}$, where NL denotes the class of problems decidable by nondeterministic logspace algorithms, and UL the class of problems decidable by unambiguous logspace algorithms. The question is equivalent to obtaining a UL-algorithm for the NL-complete problem of Reachability: Given a digraph G and two vertices s and t, is there a path from s to t?

It is known that $\mathsf{NL} \subseteq \mathsf{UL/poly}$ [RA00]. In fact, they show that NL is in a randomized version of UL, where the UL algorithm has two-way access to a random string, and may err with low probability. The underlying algorithm is unambiguous regardless of the choice of random string. For this reason, a common non-uniform hardness assumption for space-bounded computations ($\mathsf{SPACE}[n] \notin \text{io-}\mathsf{SIZE}[2^{\epsilon \cdot n}]$ for some constant $\epsilon > 0$) implies that $\mathsf{NL} \subseteq \mathsf{UL}$ [ARZ99]. The way randomness is used in these works is via the Isolation Lemma [MVV87]: For any set system F over a finite universe U, a random assignment of integer weights in the range $\{1, \ldots, 2 \cdot |U|\}$ to the elements in U makes the set of minimum weight in F unique with high probability. In the context of space-bounded isolation, U may refer to the set of edges or the set of vertices for the Reachability instances, and the set F refers to the set of paths from s to t. Notice, however, that the isolation only guarantees, with high probability, that the minimum-weight path between s and t is unique, whereas we wish that there either is a unique path or none. To bridge this gap, Reinhardt and Allender [RA00] develop two unambiguous logspace algorithms: One for testing whether a weighted graph is *min-unique*, i.e., whether there exists a unique minimum-weight path between every pair of reachable vertices, and one for determining whether there exists a path from s to t in a min-unique graph.

There are also unconditional results on space-bounded isolation. Van Melkebeek and Prakriya proved that $\mathsf{NL} \subseteq \mathsf{UL}^{1.5}$, that is, that there exists an unambiguous algorithm that decides Reachability while using $O(\log^{1.5} n)$ space [vMP19]. In fact, they show the stronger result that there exists an unambiguous algorithm for Reachability that runs in polynomial time and $O(\log^{1.5} n)$. Their result works by constructing a pseudorandom weight-assignment generator, an algorithm that, on input 1^n, outputs a sequence of weight functions such that for every instance G of Reachability of description length n, there exists at least one weight function that is *min-isolating* for G. In fact, constructing a suitable *targeted* weight-assignment generator is equivalent to the inclusion $\mathsf{NL} \subseteq \mathsf{UL}$ [PTV12]. By a suitable targeted weight-assignment generator, we mean an unambiguous logspace algo-

rithm that, on input an instance (G, s, t) of Reachability, outputs a weight function that is min-isolating for G.

Catalytic algorithms. A catalytic algorithm has access to a limited amount of regular read-write memory, while also having access to a potentially much larger catalytic memory that can be modified, but must be restored to its initial configuration when the computation ends [BCK+14]. A common setting is when the catalytic algorithm has $O(\log n)$ bits of regular memory and $\mathsf{poly}(n)$ bits of catalytic memory, which is captured by the complexity class CL. In the nondeterministic setting, the catalytic algorithm must restore the catalytic tape on *all* computation paths.

As previously mentioned, isolation is also interesting in the setting of catalytic computation, and the main open problem is whether CNL, the class of problems decided by nondeterministic logspace catalytic algorithms, is in CUL, the catalytic logspace analogue of UL. Until this work, as far as we know, the only result surrounding isolation in the catalytic setting is that under the same derandomization assumption that implies $\mathsf{NL} = \mathsf{UL}$ ($\mathsf{SPACE}[n] \notin \text{io-}\mathsf{SIZE}[2^{\epsilon \cdot n}]$ for some constant $\epsilon > 0$), it holds that $\mathsf{CNL} = \mathsf{CUL}$ [GJS+19]. Whereas in the logspace setting it is known that $\mathsf{NL} = \mathsf{coNL}$ [Imm88; Sze88], the analogous result in the catalytic setting, $\mathsf{CNL} = \mathsf{coCNL}$, is only known under the same derandomization assumption [BKL+17].

Our results. Our first contribution is a refutation and compression condition under which space-bounded isolation can be achieved.

Theorem 5.1. *There exists a universal constant c such that the following holds for all $\epsilon \in (0, 1)$. If there exists a refuter computable in FUL for the identity function against $\mathsf{USPCOMP}[(c/\epsilon) \cdot \log n, n^{\epsilon}]$, then $\mathsf{NL} = \mathsf{UL}$.*

Where $\mathsf{USPCOMP}[s(n), \gamma(n)]$ denotes the class of bottleneck algorithms (see Section 4.3.1) with compression length $\gamma(n)$ and space bound $s(n)$, where the decompression phase is an

unambiguous algorithm.

We remark that the hypothesis of Theorem 5.1 follows from the non-uniform hardness assumption from which space-bounded isolation is known to follow, and is seemingly weaker. We also consider it to be more natural, since it concludes space-bounded isolation from a constructive hardness assumption against space-bounded unambiguous algorithms.

Proposition 5.2. *If* $\mathsf{SPACE}[n] \notin \text{io-}\mathsf{SIZE}[2^{\epsilon \cdot n}]$ *for some constant* $\epsilon > 0$*, then the hypothesis of Theorem 5.1 is true.*

We also show that the hardness assumption of Theorem 5.1 is equivalent to a derandomization task surrounding the search version of unambiguous algorithms.

Theorem 5.3. *The following are equivalent:*

1. *For a a universal constant* c *and some* $\epsilon \in (0,1)$*, there exists a refuter computable in* FUL *for the identity function against* $\mathsf{USPCOMP}[(c/\epsilon) \cdot \log n, n^{\epsilon}]$*.*
2. *There exists a targeted pseudorandom generator for* $(\mathsf{UL} \cap \mathsf{coUL})$ *computable in* FUL*.*
3. $\mathsf{Search\text{-}prBP} \cdot (\mathsf{UL} \cap \mathsf{coUL}) \subseteq \mathsf{Search\text{-}pr}(\mathsf{UL} \cap \mathsf{coUL})$*.*

Here, $\mathsf{Search\text{-}prBP} \cdot (\mathsf{UL} \cap \mathsf{coUL})$ and $\mathsf{Search\text{-}pr}(\mathsf{UL} \cap \mathsf{coUL})$ are search-problem versions of the promise-problem classes $\mathsf{prBP} \cdot (\mathsf{UL} \cap \mathsf{coUL})$ and $\mathsf{pr}(\mathsf{UL} \cap \mathsf{coUL})$, respectively. We refer the reader to Section 5.4.1 for their definitions.

In the catalytic setting, we show completely analogous results to those of the space-bounded setting. In particular, we show how to obtain isolation from weaker assumptions than previously known [GJS+19].

Theorem 5.4. *There exists a universal constant* c *such that the following holds for all* $\epsilon \in (0,1)$*. If there exists a refuter computable in* FCUL *for the identity function against* $\mathsf{CUSPCOMP}[(c/\epsilon) \cdot \log n, n^{\epsilon}]$*, then* $\mathsf{CNL} = \mathsf{CUL}$*.*

$\mathsf{CUSPCOMP}[s(n), \gamma(n)]$ denotes the class of bottleneck algorithms with space bound $s(n)$ and length compression $\gamma(n)$, where the decompression phase is an unambiguous catalytic algorithm.

The derandomization/refutation equivalence also extends to the catalytic setting, and we show a different, more technical condition under which isolation follows (Theorem 5.21).

Finally, we show that a probabilistic version of the class NL, similar in spirit to the class AM that is the focus of this dissertation, is in CL, i.e., every problem in the class admits a logarithmic space catalytic algorithm.

Theorem 5.5. $\mathsf{BPNL} \subseteq \mathsf{CL}$.

The class BPNL is the class of problems admitting the following type of Arthur-Merlin protocol: On input x, Arthur first selects a random string r, but does not read it. Merlin then replies with a message y. Finally, Arthur runs a logarithmic space computation that has two-way read only access to x and one-way read-only access to (r, y). We remark that similar classes were studied before (see [Con93]), however, among other differences, the classes in these works allow for an unbounded number of rounds of communication between Arthur and Merlin.

Organization. In Section 5.2, we present an overview of the ideas underlying our results. In Section 5.3, we present definitions, notation, and other preliminaries. In Section 5.4, we establish our conditional space-bounded isolation result (Theorem 5.13). We establish the equivalence of Theorem 5.3 in Section 5.4.1. In Section 5.5, we establish our results in the catalytic setting (Theorems 5.4 and 5.5).

5.2 Technical overview

Our refutation-based results follow the techniques that we developed in Chapter 4, and make use of recent space-efficient instantiations of the Nisan-Wigderson generator [NW94] due to

Doron and Tell [DT23].

Following Reinhardt and Allender [RA00], let UReach be an unambiguous algorithm that, on input an instance (G, s, t) of Reachability and a random string r of polynomial length in $|(G, s, t)|$ (representing a weight function w), runs the algorithms that check whether w is min-isolating for G and, if it is, outputs whether there is a path between s and t in G. By the Isolation Lemma, for each fixed (G, s, t), UReach accepts most of its random inputs. Our goal is to derandomize UReach to show that $\mathsf{NL} = \mathsf{UL}$.

Let (G, R) be the pair of generator/reconstructor due to Doron and Tell [DT23]. Fix a basis string z and a potential distinguisher D. Then one of the following must hold:

1. $G(z)$ fools D.
2. $R^D(z)$ first compresses z to a smaller string, say of length $|z|^{1/2}$, and then restores z from this compressed representation with high probability.

Moreover, both G and R run in space $O(\log |z|)$.

As with our other results based on refutation, the idea for derandomizing UReach is to use the refuter to obtain a string z such that item 2 above fails, in which case the output for the corresponding generator must be pseudorandom for $\text{UReach}(G, s, t; \cdot)$ (item 1 holds). For the equivalences, we cast the refutation problem as a probabilistic search problem, which can be derandomized under the refutation assumption in the same way. This is how we obtain Theorems 5.13 and 5.3. Theorem 5.4 is completely analogous, where we derandomize a *catalytic* unambiguous algorithm CUReach with a similar behavior to UReach.

The remaining results in the catalytic setting follow a similar strategy, though using the catalytic tape as a source of pseudorandomness (a technique named compress-or-random in [Mer23]). Let D be a randomized catalytic algorithm we wish to fool, such as CUReach. In the random case, the pseudorandom set resulting from the catalytic tape fools D, and we are done. In the compress case, the resulting set fails to fool D. Then, using a deterministic logspace variant of the Nisan-Wigderson reconstructor, due to Doron, Pyne and

Tell [DPT24], it is possible to compress a portion of the catalytic tape. By setting the parameters accordingly, this process frees up enough space to carry out a trivial derandomization for D, and we are done as well.

The proof that $\mathsf{BPNL} \subseteq \mathsf{CL}$ follows the compress-or-random technique, and develops a deterministic distinguisher-to-predictor reduction for nondeterministic read-once branching programs (NROBPs) following a similar transformation for read-once branching programs (ROBPs) due to Doron, Pyne and Tell [DPT24].

5.3 Preliminaries

5.3.1 Unambiguous computation

We define a machine model for the functional version of unambiguous computations.

Definition 5.6. Let A be a nondeterministic algorithm. We say that A is an FU algorithm if, for every input x, there is exactly one succeeding computation path for A that outputs a value, which we denote by $A(x)$. We say that A computes a language L if $A(x) = L(x)$ for every input x. FUL algorithms are defined analogously. ◂

5.3.2 Catalytic computation

We first define catalytic and nondeterministic catalytic algorithms.

Definition 5.7. A *catalytic algorithm* M is defined as an algorithm in the usual sense, i.e., a Turing machine with a read-only input tape, a write-only output tape, and a read-write work tape — with an additional read-write tape known as the *catalytic tape*. The catalytic tape is initialized to hold an arbitrary string w, and M has the restriction that for any initial setting of the catalytic tape, at the end of its computation the catalytic tape must be returned to the original state w.

A *catalytic nondeterministic algorithm* M is the nondeterministic variant of a catalytic deterministic algorithm, and similar to a nondeterministic algorithm accepts if and only if there exists a sequence of nondeterministic choices such that it accepts. Additionally, M is required to restore its catalytic tape irrespective of its nondeterministic choices. ◂

Next, we define complexity classes that are based on catalytic algorithms.

Definition 5.8. Let s be a space bound. $\mathsf{CSPACE}[s]$ is the class of languages decidable by a catalytic algorithm using s bits of regular workspace and 2^s bits of catalytic workspace. The class CL is defined as $\mathsf{CSPACE}[O(\log n)]$. $\mathsf{CNSPACE}[s]$ and CNL are the analogous classes for catalytic nondeterministic algorithms, and $\mathsf{CUSPACE}[s]$ and CUL are the analogous classes for catalytic unambiguous algorithms. ◂

The restriction to 2^s bits of catalytic space is natural as otherwise storing the information of the current position of the catalytic tape already requires more space than what the regular worktape can accommodate.

We extend the definition of FU algorithms to catalytic logspace algorithms as well, obtaining the machine class FCUL.

5.3.3 Space-bounded bottleneck algorithms

Similar to the time-bounded bottleneck algorithms of Chapter 4, we define bottleneck algorithms that have space bounds.

Definition 5.9. Let $\mathcal{A}$ be a class of promise algorithms, s a space bound and $\gamma : \mathbb{N} \to \mathbb{N}$. We let $\mathcal{A}\mathsf{SPCOMP}[s(n), \gamma(n)]$ be the class of computational problems with the following properties for some probabilistic algorithm A_{comp} and some $A_{\text{dec}} \in \mathcal{A}$. For any input $x \in \{0,1\}^*$:

- The process first runs A_{comp} on input x, yielding a string π, and then runs A_{dec} on input π.

- Each of the two phases run in space $s(|x|)$, and A_{comp} has two-way access to its random bits.
- The length of π never exceeds $\gamma(|x|)$.

◂

To obtain derandomization, we are mostly interested in refuters against space-bounded unambiguous algorithms, i.e., against classes $\mathsf{USPCOMP}[s(n), \gamma(n)]$. Algorithms in this class are pairs $(A_{\text{comp}}, A_{\text{dec}})$ where A_{dec} is a nondeterministic algorithm that always has a unique succeeding path on any input. For our catalytic space results, we also employ refuters against algorithms in classes $\mathsf{CUSPCOMP}[s(n), \gamma(n)]$, where the decompressor is a catalytic nondeterministic algorithm that always has a unique succeeding path on any input.

5.3.4 Implicit oracle access

The reconstructors that we consider all need oracle access to a distinguisher. To make sure that the overall simulations remain space-bounded once the oracles are replaced by actual space-bounded algorithms, we make sure that the reconstructors access the distinguisher in the following *implicit* way.

Definition 5.10. Let A be an algorithm and $D \subseteq \{0,1\}^*$ an oracle. Giving A implicit oracle access to D allows the algorithm to interact with D in the following way:

- A can invoke the oracle D, which passes control to the oracle.
- D can read the i-th bit of its input x, by feeding i to A. This passes control back to the algorithm A.
- A can give the oracle D a query value in $\{0, 1, \perp\}$, where $\perp$ indicates that $|x| < i$. This passes control back to D.

- D can give the algorithm A a Boolean answer value. This passes control back to the algorithm.

At any moment in time, there is at most one unresolved oracle or input query. Moreover, for any input $x \in \{0,1\}^n$ the oracle D is guaranteed to only make input queries in $[n+1]$. ◂

5.4 Space-bounded isolation

In this section, we show that the existence of unambiguous space-bounded refuters for the identity function against unambiguous space-bounded bottleneck algorithms implies space-bounded isolation.

We need the following randomized unambiguous algorithm for reachability, due to Reinhardt and Allender:

Lemma 5.11 ([RA00], see also [vMP19]). *There exists a probabilistic* FUL *algorithm, which we denote by* UReach *that, on input a digraph* $G = (V, E)$ *on* n *vertices,* $s, t \in V$*, and two-way access to a random string* r*, either outputs* $\perp$ *(indicating a "bad" random string) or succeeds and outputs a bit indicating whether there is a path from* s *to* t *in* G*.* UReach *requires* $\mathsf{poly}(n)$ *random bits and succeeds with probability at least* $2/3$*.*

Notice that, regardless whether UReach succeeds or fails there is always only a single computation path that leads to this outcome.

We also need the following variant of the Nisan-Wigderson generator/reconstructor due to Doron and Tell:

Lemma 5.12 (Follows from [DT23]). *There exist a constant* c_{NW}*, an algorithm* NW *(the generator) and a pair* A_{rec} *(the reconstructor) consisting of a probabilistic algorithm* A_{comp} *and an oracle algorithm* A_{dec} *such that for every* $z \in \{0,1\}^*$*,* $\delta \in (0,1)$ *and algorithm* $D : \{0,1\}^m \to \{0,1\}$*, where* $m = |z|^\delta$*, at least one of the following holds.*

1. $\mathrm{NW}(z, 1^m)$ $1/m$*-fools* D*.*

2. $A^D_{dec}(A_{comp}(z, 1^m))$ *outputs* z *with probability at least* $2/3$.

The construction also has the following properties:

- Compression: *On input* z *of length* n *and* 1^m*, the output of* A_{comp} *has length at most* $m^{\delta \cdot c_{\mathrm{NW}}}$.
- Space efficiency: *On input* z *of length* n *and* 1^m*,* NW*,* A_{comp} *and* A_{dec}*, ignoring the space used by the oracle calls, run in space* $(c_{\mathrm{NW}}/\delta) \cdot \log n$.
- Oracle access: A_{dec} *has implicit access to* D *(see Definition 5.10).*

In the original phrasing of the Doron-Tell result, their reconstructor outputs a TC^0 circuit of size $|z|^{\delta \cdot c_{\mathrm{NW}}}$ that makes non-adaptive queries to the distinguisher D. In our case, we take A_{comp} to be their reconstructor, and A_{dec} to be an algorithm that evaluates D on all indices to recover z, and instantiate their reconstructor with a slightly smaller value of δ so that the description for this circuit has length $|z|^{\delta \cdot c_{\mathrm{NW}}}$. In the same work, and using similar ideas as the ones underlying the inclusions $\mathsf{TC}^0 \subseteq \mathsf{NC}^1 \subseteq \mathsf{L}$, they also show how to adapt the DFS-style logspace simulation for NC^1 circuits to accommodate oracle-TC^0 circuits. As the circuit is non-adaptive, whenever the DFS-style simulation reaches an oracle gate and starts simulating D, no other oracle calls to D are made until the oracle gate is fully evaluated, resulting in the type of oracle access in our statement.

We are now ready to state and prove the main result for this section.

Theorem 5.13 (Theorem 5.1, restated). *There exists a universal constant* c *such that the following holds for all* $\epsilon \in (0, 1)$*. If there exists a refuter computable in* FUL *for the identity function against* $\mathsf{USPCOMP}[(c/\epsilon) \cdot \log n, n^\epsilon]$*, then* $\mathsf{NL} = \mathsf{UL}$.

Proof. We show that, under the assumption, there exists an unambiguous algorithm A_U that solves the Reachability problem using logarithmic space. It then follows by the NL-completeness of Reachability that $\mathsf{NL} = \mathsf{UL}$.

Let NW be the generator and $A_{\text{rec}} = (A_{\text{comp}}, A_{\text{dec}})$ be the reconstructor from Lemma 5.12. Let ϵ be the constant in the statement, and R be the refuter against $\mathsf{USPCOMP}[(c/\epsilon) \cdot \log n, n^{\epsilon}]$. On input an instance (G, s, t) of Reachability of length n, the high-level idea is as follow: First, recall that our objective is to derandomize UReach. To do so, A_U computes the output for the refuter on input $1^{n'}$ for a sufficiently large n' to be defined next and the bottleneck algorithm $(A_{\text{comp}}, A_{\text{dec}}^{\text{UReach}(G,s,t;\cdot)})$, where $(A_{\text{comp}}, A_{\text{dec}})$ is the reconstructor of Lemma 5.12. By setting n' correctly, the refuter must output z of length at least n' such that $A_{\text{dec}}^{\text{UReach}(G,s,t;\cdot)}(A_{\text{comp}}(z, 1^m))$ fails to output z with high probability, in which case NW with input z must fool $\text{UReach}(G, s, t; \cdot)$, achieving our objective. Details follow.

Let k be a constant such that $\text{UReach}(G, s, t; \cdot)$ needs $m = n^k$ random bits on inputs of length n. We set $n' = m^{1/\delta}$, for a value of δ to be defined next. Recall that, if the generator of Lemma 5.12 fails with string z, then the reconstructor compresses z to length $|z|^{\delta \cdot c_{\text{NW}}}$, where c_{NW} is a universal constant. We thus set $\delta = \epsilon / c_{\text{NW}}$, getting compression length $|z|^{\epsilon}$ as in the theorem statement. To guarantee that the refuter is successful on input $1^{n'}$ and $(A_{\text{comp}}, A_{\text{dec}}^{\text{UReach}(G,s,t;\cdot)})$, it still remains to analyze the space complexity for A_{comp} and $A_{\text{dec}}^{\text{UReach}(G,s,t;\cdot)}$ on inputs of length $\ell \geq n'$, and to argue that $A_{\text{dec}}^{\text{UReach}(G,s,t;\cdot)}$ is an FU algorithm.

By Lemma 5.12, the space complexity of A_{comp} on an input of length ℓ is $(c_{\text{NW}}/\delta) \cdot \log \ell = (c_{\text{NW}}^2/\epsilon) \cdot \log n$, which is also a space-bound for A_{dec} ignoring the oracle calls to $\text{UReach}(G, s, t; \cdot)$. We now analyze $A_{\text{dec}}^{\text{UReach}(G,s,t;\cdot)}$. Given the type of oracle access that A_{dec} has to $\text{UReach}(G, s, t; \cdot)$, its space-complexity is upper-bounded, up to a constant, by the sum of the space-complexities of A_{dec} and $\text{UReach}(G, s, t; \cdot)$. As the space complexity of $\text{UReach}(G, s, t; \cdot)$ is $O(\log n)$, and the input length for $(A_{\text{comp}}, A_{\text{dec}})$ is $\ell \geq \mathsf{poly}(n)$, there exists a constant $c > c_{\text{NW}}^2$ such that both A_{comp} and $A_{\text{dec}}^{\text{UReach}(G,s,t;\cdot)}$ run in space $(c/\epsilon) \cdot \log \ell$. Since A_{dec} is deterministic apart from the oracle calls to $\text{UReach}(G, s, t; \cdot)$, it follows that $A_{\text{dec}}^{\text{UReach}(G,s,t;\cdot)}$ is an FU algorithm.

It remains to analyze A_U. First, A_U computes the refuter on an input of length $n' + O(n) = \mathsf{poly}(n)$, where the $O(n)$ term captures the description length of $(A_{\text{comp}}, A_{\text{dec}}^{\text{UReach}(G,s,t;\cdot)})$.

Thus, this first step requires unambiguous space $O(\log n)$, and produces a string z of length $\mathsf{poly}(n)$. Then, A_U computes $\mathrm{UReach}(G, s, t; \rho)$ for each ρ output by $\mathrm{NW}(z, 1^m)$ until obtaining a success, which is guaranteed by the discussion above. Using standard space-efficient composition, and under the usual rule for composing unambiguous algorithms where the overall computation rejects if a computation path for either unambiguous algorithm rejects, the overall space-complexity for A_U is $O(\log n)$. ■

We believe that isolation is indeed equivalent to the refutation assumption of Theorem 5.13, but we were unable to prove this equivalence so far. In the next section, we present a derandomization assumption that captures this refutation task.

5.4.1 An equivalence between refutation and derandomization

Toward obtaining an equivalence, and following Definition 3.20, we introduce the search version of the class $\mathsf{prBP} \cdot (\mathsf{UL} \cap \mathsf{coUL})$.

Definition 5.14. Let R_Y and R_N be two disjoint binary relations. We say that (R_Y, R_N) is in $\mathsf{Search\text{–}prBP} \cdot (\mathsf{UL} \cap \mathsf{coUL})$ if the following two conditions hold.

1. The decisional problem represented by (R_Y, R_N) is in $\mathsf{prBP} \cdot (\mathsf{UL} \cap \mathsf{coUL})$; that is, there exists a $(\mathsf{UL} \cap \mathsf{coUL})$ algorithm V such that for every $(x, y) \in R_Y$, $\Pr_r[V(x, y; r) = 1] \geq 2/3$ and for every $(x, y) \in R_N$, $\Pr_r[V(x, y; r) = 1] \leq 1/3$.

2. There exists an FUL algorithm G such that, for every x for which $R_Y(x) \neq \varnothing$, it holds that $\Pr_r[G(x; r) \in R_Y(x)] \geq 2/3$, where $R_Y(x) = \{y \mid (x, y) \in R_Y\}$.

Both V and G are allowed two-way access to their random bits. ◂

We similarly define the search version of $\mathsf{Search\text{–}pr}(\mathsf{UL} \cap \mathsf{coUL})$, where the solution finding algorithm G and the verifier V are both $(\mathsf{UL} \cap \mathsf{coUL})$ algorithms.

First, we show that the refutation assumption of Theorem 5.13 implies derandomization for the class $\mathsf{Search\text{–}prBP} \cdot (\mathsf{UL} \cap \mathsf{coUL})$.

Lemma 5.15. *There exists a universal constant c such that the following holds for all $\epsilon \in (0,1)$. If there exists a refuter computable in* FUL *for the identity function against* $\mathsf{USPCOMP}[(c/\epsilon)\cdot\log n, n^\epsilon]$*, then* $\mathsf{Search\text{–}prBP}\cdot(\mathsf{UL}\cap\mathsf{coUL}) \subseteq \mathsf{Search\text{–}pr}(\mathsf{UL}\cap\mathsf{coUL})$.

Proof. By replacing UReach by a generic $(\mathsf{UL}\cap\mathsf{coUL})$ algorithm with two-way access to its random bits, the argument of Theorem 5.13 shows that, under the refutation assumption in the theorem statement, there exists a targeted pseudorandom generator for $(\mathsf{UL}\cap\mathsf{coUL})$ that is itself computable in FUL.

Let $(R_Y, R_N) \in \mathsf{Search\text{–}prBP}\cdot(\mathsf{UL}\cap\mathsf{coUL})$. First, notice that the existence of a targeted generator as in the previous paragraph implies that $\mathsf{prBP}\cdot(\mathsf{UL}\cap\mathsf{coUL}) \subseteq \mathsf{pr}(\mathsf{UL}\cap\mathsf{coUL})$, and thus there exists an unambiguous verifier V_{derand} for (R_Y, R_N) that runs in logarithmic space, requiring no randomness. Let G be the solution-finding algorithm for (R_Y, R_N) and fix an input $x \in \{0,1\}^n$. Note that for any r such that $V_{\text{derand}}(x, G(x;r)) = 1$, we have that $G(x;r)$ is a valid solution for the input x. Therefore, it suffices to compute the multi-set S output by the targeted PRG on input x and the algorithm $V_{\text{derand}}(x, G(x;\cdot))$, compute $V_{\text{derand}}(x, G(x;r))$ for each $r \in S$, and output $G(x;r)$ for the first r that is accepted. The entire procedure runs in unambiguous logarithmic space by standard space-efficient composition results, and thus $(R_Y, R_N) \in \mathsf{Search\text{–}pr}(\mathsf{UL}\cap\mathsf{coUL})$. ∎

We are now able to prove the equivalence of Theorem 5.3.

Proof of Theorem 5.3. That 1 $\implies$ 2 and 2 $\implies$ 3 both follow from the proof of Lemma 5.15. To see that 3 $\implies$ 1, we show that computing a refuter as in the first item is a $\mathsf{Search\text{–}prBP}\cdot(\mathsf{UL}\cap\mathsf{coUL})$ problem. The argument is very similar to that of Theorem 4.21. We include the details for completeness.

Consider an input $(1^n, A_{\text{comp}}, A_{\text{dec}})$ for the refutation task, and fix a value of $\epsilon \in (0,1)$. Let A'_{comp} be the version of A_{comp} with a $(c/\epsilon)\cdot\log n$ space bound, and similarly let A'_{dec} be the version of A_{dec} that has a $(c/\epsilon)\cdot\log n$ space bound and outputs 0^n in case its input has length greater than n^ϵ. Define (R_Y, R_N) as follows:

- $(1^n, A_{\text{comp}}, A_{\text{dec}}, z) \in R_Y$ if and only if $|z| \geq n$ and $\Pr[(A'_{\text{comp}}, A'_{\text{dec}}(z)) = z] \leq 1/3$.
- $(1^n, A_{\text{comp}}, P_{\text{dec}}, z) \in R_N$ if and only if $|z| \geq n$ and $\Pr[(A'_{\text{comp}}, A'_{\text{dec}}(z)) = z] \geq 2/3$.

We first show how to obtain a counterexample with high probability. Assume that $(A'_{\text{comp}}, A'_{\text{dec}})$ computes the identity function with probability at least $1/3$ 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 $A'_{\text{dec}}(A'_{\text{comp}}(z; r_1); r_2)$ outputs z with probability at least $1/3$ (over r_2). If we let $\pi_z = A'_{\text{comp}}(z; r_1)$ and fix $(A'_{\text{comp}}, A'_{\text{dec}})$, we can describe z as one of the only three possible strings that $A'_{\text{dec}}(\pi_z)$ outputs with probability at least $1/3$. 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 $2^{n_0^\epsilon + c + 1}/2^{n_0} \leq 1/3$. In this case, it suffices for the solution-finding algorithm to output a random string of length ℓ, which will be a good counterexample with probability at least $2/3$.

As for verifying a solution, on input $(1^n, A_{\text{comp}}, A_{\text{dec}})$ and a candidate counterexample z, it suffices to approximate, via sampling up to error 0.1 and with confidence 0.9, the probability that $A'_{\text{dec}}(A'_{\text{comp}}(z)) = z$, and accept if it is lower than 0.5. The process only requires a constant number of samples/simulations, and thus can be performed in $(\mathsf{UL} \cap \mathsf{coUL})$. ∎

We finish this section by observing how the previously-known assumption for isolation is stronger than the assumption of Theorem 5.13.

Proof (Sketch) of Proposition 5.2. The assumption of Theorem 5.13 is equivalent to the existence of targeted pseudorandom generators for unambiguous logspace computations that are themselves computable in unambiguous logspace, while the assumption $\mathsf{SPACE}[n] \notin \text{io-}\mathsf{SIZE}[2^{\epsilon \cdot n}]$ for some $\epsilon > 0$ is equivalent the existence of regular (oblivious) pseudorandom generators for polynomial-size circuits, computable in logarithmic space. As oblivious PRGs are a specific type of targeted PRG, and $\mathsf{UL} \subseteq \mathsf{P/poly}$, the result follows. ∎

5.5 Space-bounded catalytic computation

In this section, we explore the previously-developed techniques relating refutation and compression to the catalytic setting.

We need the following randomized unambiguous algorithm for nondeterministic catalytic computations, due to Gupta, Jain, Sharma and Tewari.

Lemma 5.16 ([GJS+19]). *There exists a probabilistic* FCUL *algorithm, denoted* CUReach *that, on input the description of a* CNL *algorithm M, $x \in \{0,1\}^n$, catalytic tape contents $w \in \{0,1\}^{\mathsf{poly}(n)}$, and two-way access to a random string r, either outputs $\perp$ (indicating a "bad" random string) or succeeds and outputs $M(x,w)$. The machine requires $\mathsf{poly}(n)$ random bits and succeeds with probability at least $2/3$.*

We remark that the hidden constants in the time, catalytic space, and randomness complexity of CUReach depend on the particular machine M it receives as input.

Following the same strategy as with Theorem 5.13, it is possible to conclude that CNL = CUL from the existence of a refuter against bottleneck *catalytic* unambiguous algorithms.

Theorem 5.17 (Theorem 5.4, restated). *There exists a universal constant c such that the following holds. If for a sufficiently small constant $\epsilon \in (0,1)$ there exists an* FCUL *unambiguous refuter for the identity function against* $\mathsf{CUSPCOMP}[(c/\epsilon)\cdot \log n, n^{\epsilon}]$*, then* CNL = CUL.

Proof (sketch). The proof is almost identical to that of Theorem 5.13, though this time the potential distinguisher is an FCUL algorithm (the algorithm CUReach of Lemma 5.16). Given the way A_{dec} in Lemma 5.12 accesses its oracle, and the fact that the same strategy for achieving space-efficient composition works for catalytic computations, it follows similarly that $A_{\mathrm{dec}}^{\mathrm{CUReach}}$ is an FCUL algorithm. The same strategy is used to guarantee that the final simulation for a CNL language runs in $(\mathsf{CUL} \cap \mathsf{coCUL})$. ∎

Also similar to the regular space-bounded setting, we can obtain an equivalence with derandomizing the search version of $\mathsf{prBP}\cdot(\mathsf{CUL}\cap\mathsf{coCUL})$. We refrain from formally presenting this equivalence as it does not introduce any new ideas. We do mention, however, that the refutation assumption of Theorem 5.4, while incomparable to that of Theorem 5.13 (since it has a more relaxed requirement for the computability of the refuter while also strengthening the class of algorithms to be refuted), is also weaker than the previously-known assumption from which $\mathsf{CNL} = \mathsf{CUL}$ follows, that of linear-exponential circuit lower bounds for $\mathsf{SPACE}[n]$ [GJS+19]. This is the case because since $\mathsf{CUL} \subseteq \mathsf{CNL} \subseteq \mathsf{ZPP}$ [BKL+17], a pseudorandom generator for polynomial-size circuits suffices to derandomize the search version of $\mathsf{prBP} \cdot (\mathsf{CUL} \cap \mathsf{coCUL})$ that is equivalent to the refutation task.

5.5.1 Isolation via the compress-or-random framework

In this section, we present another condition under which $\mathsf{CNL} = \mathsf{CUL}$. Toward developing this result, we formally introduce the notion of a *predictor*, we used before in Section 3.4. Many pseudorandom generator constructions, and in particular the Nisan-Wigderson construction [NW94], which we use heavily in this chapter, rely on transforming a distinguisher for the generator into a previous bit predictor.[1]

We have already defined to notion of δ-fooling in Section 3.5.5. We now define the notion of δ-predicting.

Definition 5.18. Let G be a distribution over $\{0,1\}^m$, $\delta \in [0, 1/2]$ and P an algorithm/oracle. We say that P is a δ-previous bit predictor for G if there exists $i \in \{1, \ldots, m\}$ such that

$$\Pr_{r \leftarrow G}[P(r_{>i}) = r_i] \geq \frac{1}{2} + \delta,$$

that is, P, when receiving the last $m - i$ bits of a random sample r of G, outputs the i-th bit of r with probability at least $1/2 + \delta$. ◂

[1] For the Nisan-Wigderson construction, it does not matter whether the transformation is to a previous-bit predictor or a next-bit predictor.

When considering a predictor for a pseudorandom generator G, the advantage of the predictor is measured with respect to a randomly selected element in the multi-set output by G.

We also introduce the notion of a distinguisher-to-predictor transformation.

Definition 5.19. We say that a class of algorithms $\mathcal{A}$ has a distinguisher-to-predictor transformation if there is a meta-algorithm that, given $A(\cdot,\cdot) \in \mathcal{A}$, an input x of length n and 1^m, outputs a collection of $\mathcal{A}$ algorithms (that also get input access to x) $P_1, \ldots, P_{\mathsf{poly}(m,n)}$ such that for every distribution G over $\{0,1\}^m$, one of the following occurs:

1. G $(1/m)$-fools $A(x,\cdot)$.
2. There is i such that $P_i(x,\cdot)$ is a $(1/m^2)$-previous-bit-predictor for G.

◂

The choice of $1/m^2$ for the advantage for the predictor stems from the hybrid argument which, starting from a distinguisher on m input bits that is not $1/m$-fooled by G, probabilistically constructs a $1/m^2$ previous-bit predictor.

Finally, we introduce another variant of the Nisan-Wigderon generator/reconstructor, due to Doron, Pyne and Tell (though stated in this format by Cook, Li, Mertz and Pyne):

Lemma 5.20 ([CLM+24], following [DPT24]). *There exist universal constants $c_{\mathrm{NW}} > 1$ and $c_{\mathrm{NW}} > \gamma > 0$ such that the following holds. There exists algorithms* NW*, A_{erase} and $A_{recover}$ such that for any $m \in \mathbb{N}$ and $z \in \{0,1\}^{m^{c_{\mathrm{NW}}}}$, we have the following:*

1. Efficiency. $\mathrm{NW}(z)$ *runs in space $O(\log m)$ and outputs a set of m-bit strings.*

2. Deterministic Reconstruction. *A_{erase} and $A_{recover}$ describe a reconstruction process that act as follows:*

- A_{erase}, *given oracle access to* z *and implicit oracle access to a* $(1/m^2)$*-previous bit predictor* P *for* $\mathrm{NW}(z)$, *returns* $h \in \{0,1\}^{O(\log m)}$ *and the endpoints of an interval* $I \subseteq [m^{c_{\mathrm{NW}}}]$ *of length* m^{γ}.
- $A_{recover}$, *given implicit oracle access to* P *and oracle access to* $\tilde{z}$ *such that* $\tilde{z}$ *agrees with* z *on every position outside of* I, *satisfies for every* $j \in I$:

$$A_{recover}^{\tilde{z},P}(h,j) = z_j.$$

We are now ready to state and prove the other condition under which we may conclude that $\mathsf{CNL} = \mathsf{CUL}$.

Theorem 5.21. *Assume that there exists a distinguisher-to-predictor transformation for* $(\mathsf{CUL} \cap \mathsf{coCUL})$ *computable in* FCUL, *then* $\mathsf{CNL} = \mathsf{CUL}$.

Proof. Let $L \in \mathsf{CNL}$ and M a nondeterministic catalytic machine that decides L in catalytic space $c \cdot \log n$ for some constant c. Let CUReach be the algorithm from Lemma 5.16.

The high-level strategy is to use an additional catalytic tape v of sufficiently large length $\mathsf{poly}(n)$ as a basis for the generator of Lemma 5.20 to attempt to hit the algorithm CUReach with regular inputs M and x and catalytic tape w. If that fails, then we run the distinguisher-to-predictor reduction for CUReach, obtaining a list of candidate predictors with which to run the reconstructor for Lemma 5.20. Because the compression achieved by the reconstructor allows for describing a significant interval of the catalytic tape via the remaining contents, the algorithm can test the reconstruction with each predictor until it finds one that works. When it finds one that works, it erases the interval and uses the space that was freed to run a trivial polynomial-space algorithm for L, using the fact that $\mathsf{CNL} \subseteq \mathsf{ZPP} \subseteq \mathsf{PSPACE}$. Details follow.

Let c' be a sufficiently large constant such that L is decidable in space $n^{c'}$ and CUReach requires at most $n^{c'}$ random bits on input the algorithm M and a string of length n. Let $x \in \{0,1\}^n$ be an input and $w \in \{0,1\}^{n^c}$ be the initial contents for the catalytic tape of M.

The unambiguous catalytic algorithm A_U for L has access to a catalytic tape consisting of (w, v), where v has length $|v| = n^{c' \cdot c_{\mathrm{NW}}/\gamma}$, for the constants c_{NW} and γ of Lemma 5.20.

The operation of A_U is as follows: First, it computes the multi-set S output by generator NW from Lemma 5.20 instantiated with the string v. Then, for each string r in S, it computes the value of $\mathrm{CUReach}(x, w; r)$. If any of the executions succeeds, then it outputs whatever CUReach does for that value of r. In this case, we are guaranteed that the output equals $M(x, w)$. The computation until this point runs in $(\mathsf{CUL} \cap \mathsf{coCUL})$ using standard space-efficient composition.

In case none of the executions succeed, A_U runs the assumed distinguisher-to-predictor reduction for $\mathrm{CUReach}(x, w; \cdot)$, obtaining a list of candidate predictors $P_1, \ldots, P_{\mathsf{poly}(n)}$. Let A_{erase} and A_{recover} be the algorithms from Lemma 5.20. For each predictor P_i, U performs the following actions: First, it computes $A_{\mathrm{erase}}^{v,P_i}$, obtaining a string h_i of length $O(\log n)$ and the endpoints of an interval I_i of length $|v|^{\gamma/c_{\mathrm{NW}}} = n^{c'}$. Then, it checks whether $A_{\mathrm{recover}}^{v,P_i}(h_i, j) = v_j$ for all $j \in I_i$. Because $\mathrm{CUReach}(x, w; \cdot)$ succeeds on most random inputs yet fails on every output of $\mathrm{NW}(v)$, and given the assumed distinguisher-to-predictor reduction, there must be some P_i such that $A_{\mathrm{recover}}^{v,P_i}(h_i, j) = v_j$ for all $j \in I_i$. For the first i such that this test succeeds, A_U sets the part of v indexed by I_i as $0^{n^{c'}}$ and uses it as a regular read-write memory to run the $n^{c'}$-space algorithm for L, obtaining the value of $M(x, w)$ and finishing with catalytic tape (w, v'), where v' agrees with v on every bit except for the ones in interval I_i. It then restores v by running $A_{\mathrm{recover}}^{v',P_i}(h_i, j)$ for every $j \in I_i$ and outputs $M(x, w)$. Because of the way A_{recover} accesses its oracle and just like in the proof of Theorem 5.13, $A_{\mathrm{recover}}^{v',P_i}(\cdot)$ is an FCUL algorithm. By reusing the space for storing each string h_i and interval I_i, and computing the description for each predictor bit-by-bit space-efficiently, the entire simulation runs in $(\mathsf{CUL} \cap \mathsf{coCUL})$. ∎

We remark that the standard randomized distinguisher-to-predictor reduction implies that, for a distribution computable in FCUL, computing a distinguisher-to-prediction reduction can be solved in $\mathsf{Search\text{-}pr}(\mathsf{CUL} \cap \mathsf{coCUL})$. As this relaxation suffices to establish

Theorem 5.21, it follows that the hypothesis of Theorem 5.21 is no stronger than that of Theorem 5.4.

5.5.2 Probabilistic NL is in CL

We conclude with another application of the compress-or-random framework together with distinguisher-to-predictor reductions, by showing the inclusion $\mathsf{BPNL} \subseteq \mathsf{CL}$. We start by defining the class BPNL.

Definition 5.22. BPNL contains languages $L \in \{0,1\}^*$ for which there exists a deterministic algorithm M running in space $O(\log n)$ such that:

$$x \in L \implies \Pr_r[\exists y \ M(x,y;r) = 1] \geq 2/3$$
$$x \notin L \implies \Pr_r[\exists y \ M(x,y;r) = 1] \leq 1/3.$$

M has two-way read access to x, and one-way read access to both y and r. It also first reads the entirety of r and only after that starts reading y. ◂

The class BPNL differs from the randomized logspace classes considered until now such as $\mathsf{BP} \cdot (\mathsf{UL} \cap \mathsf{coUL})$ because the underlying machines for problems in BPNL only have one-way access to their random bits, while the other classes considered all require two-way access to the random bits. Note a direct comparison with the class AM that was a major focus for this dissertation: We can view the process underlying a BPNL algorithm as first having Arthur select a random string (without looking at it) and then Merlin providing a reply. In the end, Arthur reads his random string, followed by Merlin's response, and then makes a decision, all the while being limited to logarithmic space. By enumerating the polynomially-many random and nondeterministic guesses and simulating the process, we remark that $\mathsf{BPNL} \subseteq \mathsf{PSPACE}$ (though our result implies, in particular, that $\mathsf{BPNL} \subseteq \mathsf{ZPP}$).

It is worth mentioning that even slight modifications to BPNL make it potentially much larger. For example, having Merlin go first in the interactive proof definition above leads to the class NP [Lip90].

The computation of a BPNL algorithm M on input x can be described via a nondeterministic read-once branching program (NROBP) of polynomial width, which we define next.

Definition 5.23. A nondeterministic read-once branching program (NROBP) of width w is specified by an initial state $v_s \in [w]$, an accepting state $v_a \in [w]$ and a sequence of transition functions $B_i : [w] \times \{0,1\} \to [w]$ for $i \in [2m]$. The NROBP defines a function $B : \{0,1\}^{2m} \to \{0,1\}$ as follows. On input (r, y), start at v_s. Then for $i = 1, \ldots, n$, read the symbol r_i and transition to the state $v_i = B_i(v_{i-1}, r_i)$. After that, do the same for y, ending at some state in $[w]$. We generally only consider r as the actual input for the NROBP B, and say that it accepts r, i.e., $B(r) = 1$, if and only if there exists y such that the process above with (r, y) ends at state v_a. ◂

The conversion from a BPNL algorithm M on input $x \in \{0,1\}^n$ into an NROBP can be performed in deterministic space $O(\log n)$ by following the standard transformation for logspace algorithms into Read-once branching programs (ROBPs), where the hidden constant depends on M. We develop a distinguisher-to-predictor transformation for NROBPs along the lines of the same transformation for ROBPs by Doron, Pyne and Tell [DPT24]. To state and prove this result, we first introduce subprograms for NROBPs

Definition 5.24. For an NROBP $B : \{0,1\}^{2m} \to \{0,1\}$ of length n and width w, let $B_{i,j}$ be the subprogram of length $m - i$ and width w defined as follows. We let $B_{i,j}$ be B with the first i layers removed, and vertex j in layer i is marked as the new start vertex. Note that $B_{i,j}$ can be described with $\log(nw)$ bits given B, and can be constructed in logspace given B. ◂

We now show that NROBPs admit a logspace distinguisher-to-predictor reduction.

Lemma 5.25. *Given an NROBP B of length $2m$ and width w, for every $i \in [m]$ (representing an index into the random string it reads), $j \in [w]$, and $b \in \{0,1\}$, let $P_{i,j,b} : \{0,1\}^{m-i} \to \{0,1\}$*

be defined as $P_{i,j,b}(r) = B_{i,j}(r) \oplus b$. *Then for every* $\delta > 0$, *for every NROBP* B, *for every distribution* G *over* $\{0,1\}^m$, *at least one of two events occurs:*

1. G δ*-fools* B, *or*

2. *there is* i, j, b *such that* $P_{i,j,b}$ *is a* δ/m*-previous-bit predictor for* G.

Proof. We assume that the first item does not occur and show that the second item must then hold. For $i \in \{0, \ldots, m\}$, let Z_i denote the hybrid distribution where the first i bits are uniform and the remaining ones are sampled according to G. By assumption, we have $|\mathbb{E}[B(G)] - \mathbb{E}[B(U_m)]| > \delta$, where U_m denotes the uniform distribution on m bits. By the standard transformation from distinguishability to predictability, there is $z \in \{0,1\}^i$ and $b \in \{0,1\}$ such that

$$\Pr_{r \leftarrow G}[B(z \circ r_{>i}) \oplus b = r_i] > \frac{1}{2} + \frac{\delta}{m}.$$

But observe that $B(z \circ r_{>i}) = B_{i,j}(r_{>i})$ for some $j \in [w]$, as fixing the first i bits to z is equivalent to starting the computation from the state j that B reaches after processing z. This concludes the proof since $P_{i,j,b}(r) = B_{i,j}(r) \oplus b = B(z \circ r_{>i}) \oplus b$. ∎

We remark that, since $\mathsf{NL} = \mathsf{coNL}$, the construction of Lemma 5.25 can be evaluated in NL even if $b = 1$, which requires negating the computation. Now are ready to show that $\mathsf{BPNL} \subseteq \mathsf{CL}$.

Proof of Theorem 5.5. The proof is similar to that of Theorem 5.21, so we focus on the differences. Let $L \in \mathsf{BPNL}$ and M be a logspace nondeterministic algorithm witnessing this containment that requires space $c \cdot \log n$ and $m = n^c$ random/nondeterministic bits. We describe a catalytic logspace algorithm A_C that decides L. A_C receives as input $x \in \{0,1\}^n$ and has access to a catalytic tape (w, v) of sufficiently large length $\mathsf{poly}(n)$.

As in Theorem 5.21, A_C computes the generator NW of Lemma 5.20 with input v. However, before trying to use $\mathrm{NW}(v)$ to derandomize $M(x;\cdot)$, A_C runs the predictor-to-distinguisher transformation underlying Lemma 5.25, obtaining a list of candidate predictors.

In more detail, note that the computation of $M(x;\cdot)$ can be captured by an NROBP B of length $2m$ and width m. A_C can then enumerate values of $i \in [m]$, $j \in [m]$ and $b \in \{0,1\}$, where each one describes (with $O(\log m) = O(\log n)$ bits) a candidate predictor $P_{i,j,b}$ out of $\mathsf{poly}(n)$ many. Then A_C checks, in the same way as in Theorem 5.21, whether there is a "good" predictor by checking whether the reconstructor $(A_{\text{erase}}, A_{\text{recover}})$ of Lemma 5.20 achieves the compression it is supposed to in case some $P_{i,j,b}$ is a previous-bit predictor with sufficient advantage. To evaluate the $P'_{i,j,b}s$, we use the fact that $\mathsf{NL} \subseteq \mathsf{CL}$ [BCK+14], and have A_C use the w portion of its catalytic tape as the catalytic tape for the NL simulation. If any of the predictors achieves enough compression, then we can free enough space to run a trivial space simulation for L, as in Theorem 5.21 and using the fact that $\mathsf{BPNL} \subseteq \mathsf{PSPACE}$. Otherwise, it must be the case that $\mathrm{NW}(v)$ $1/m$-fools $M(x;\cdot)$, in which case A_C can compute $\mathrm{NW}(v)$, use each string in the result multi-set as the randomness for $M(x;\cdot)$ and output the majority answer, again using the fact that $\mathsf{NL} \subseteq \mathsf{CL}$ to simulate M.

Correctness follows from the discussion above. That A_C requires only regular workspace $O(\log n)$ follows from standard space-efficient composition results. ■

5.6 Further research

In this chapter, we explored applications of the compression and refutation ideas of the previous chapters to the settings of space-bounded isolation and catalytic computation.

A recent result in the setting of catalytic computation unconditionally derandomizes the class of probabilistic logspace catalytic algorithms [CLM+24] by further expanding on some of the same ideas that are present in this chapter. We believe that the new techniques introduced by [CLM+24] can be leveraged to obtain unconditional results for space-bounded catalytic computations, such as showing that $\mathsf{CNL} = \mathsf{coCNL}$ and $\mathsf{CNL} = \mathsf{CUL}$.

Bibliography

[Adl78] Leonard Adleman. Two theorems on random polynomial time. In *Symposium on Foundations of Computer Science (FOCS)*, pages 75–83, 1978.

[AGH+11] 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.

[ARZ99] Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting uniform and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59(2):164–181, 1999.

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

[BCK+14] Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: Catalytic space. In *Symposium on Theory of Computing (STOC)*, pages 857–866, 2014.

[BFN+93] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3(4):307–318, 1993.

[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.

[BGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM Journal on Computing*, 36(4):889–974, 2006.

[BKL+17] Harry Buhrman, Michal Koucký, Bruno Loff, and Florian Speelman. Catalytic space: Non-determinism and hierarchy. *Theory of Computing Systems*, 62(1):116–135, 2017.

[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.

[BR94] Mihir Bellare and John Rompel. Randomness-efficient oblivious sampling. In *Symposium on Foundations of Computer Science (FOCS)*, pages 276–287, 1994.

[CJS+24] Lijie Chen, Ce Jin, Rahul Santhanam, and Ryan Williams. Constructive separations and their consequences. *TheoretiCS*, 3(3):1–41, 2024.

[CLM+24] James Cook, Jiatu Li, Ian Mertz, and Edward Pyne. The structure of catalytic space: Capturing randomness and time via compression. Technical report TR24-106, Electronic Colloquium on Computational Complexity (ECCC), 2024.

[CLO+23] Lijie Chen, Zhenjian Lu, Igor C. Oliveira, Hanlin Ren, and Rahul Santhanam. Polynomial-time pseudodeterministic construction of primes. In *Symposium on Foundations of Computer Science (FOCS)*, pages 1261–1270, 2023.

[Con93] Anne Condon. *The complexity of space bounded interactive proof systems*. In *Complexity Theory: Current Research*. 1993, pages 147–189.

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

[CRT+20] Lijie Chen, Ron D. Rothblum, Roei Tell, and Eylon Yogev. On exponential-time hypotheses, derandomization, and circuit lower bounds. In *Symposium on Foundations of Computer Science (FOCS)*, pages 13–23, 2020.

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

[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.

[DPT24] Dean Doron, Edward Pyne, and Roei Tell. Opening up the distinguisher: A hardness to randomness approach for BPL=L that uses properties of BPL. In *Symposium on Theory of Computing (STOC)*, pages 2039–2049, 2024.

[DT23] Dean Doron and Roei Tell. Derandomization with minimal memory footprint. In *Computational Complexity Conference (CCC)*, 11:1–11:15, 2023.

[FF93] Joan Feigenbaum and Lance Fortnow. Random-self-reducibility of complete sets. *SIAM Journal on Computing*, 22(5):994–1005, 1993.

[FGM+89] Martin Fürer, Oded Goldreich, Yishay Mansour, Michael Sipser, and Stathis Zachos. On completeness and soundness in interactive proof systems. *Advances in Computational Research*, 5:429–442, 1989.

[GJS+19] Chetan Gupta, Rahul Jain, Vimal Raj Sharma, and Raghunath Tewari. Unambiguous catalytic computation. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 16:1–16:13, 2019.

[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. 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. 2020.

[GS86] Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *Symposium on Theory of Computing (STOC)*, pages 59–68, 1986.

[GST03] 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.

[Imm88] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.

[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.

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

[Jus76] Jørn Justesen. On the complexity of decoding Reed-Solomon codes. *Transactions on Information Theory*, 22(2):237–238, 1976.

[Kor22a] Oliver Korten. Derandomization from time-space tradeoffs. In *Computational Complexity Conference (CCC)*, 37:1–37:26, 2022.

[Kor22b] Oliver Korten. The hardest explicit construction. In *Symposium on Foundations of Computer Science (FOCS)*, pages 433–444, 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.

[Lip90] Richard J. Lipton. Efficient checking of computations. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 207–215, 1990.

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

[LP23] Yanyi Liu and Rafael Pass. Leakage-resilient hardness vs randomness. In *Computational Complexity Conference (CCC)*, 32:1–32:20, 2023.

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

[Mer23] Ian Mertz. Reusing space: Techniques and open problems. *Bulletin of EATCS*, 141:57–106, 2023.

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

[MVV87] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[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.

[PTV12] Aduri Pavan, Raghunath Tewari, and N. Variyam Vinodchandran. On the power of unambiguity in log-space. *Computational Complexity*, 21(4):643–670, 2012.

[RA00] Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM Journal on Computing*, 29(4):1118–1131, 2000.

[Ram20] Chandrasekaran Ramya. Recent progress on matrix rigidity – A survey. *Computing Research Repository (CoRR)*, abs/2009.09460, 2020.

[RR97] Alexander A Razborov and Steven Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1):24–35, 1997.

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

[Sho88] Victor Shoup. New algorithms for finding irreducible polynomials over finite fields. In *Symposium on Foundations of Computer Science (FOCS)*, pages 283–290, 1988.

[Sho92] Victor Shoup. Searching for primitive roots in finite fields. *Mathematics of Computation*, 58(197):369–380, 1992.

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

[STV01] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.

[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.

[Sud97] Madhu Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, 1997.

[Sze88] Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.

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

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

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

[vMM23a] Dieter van Melkebeek and Nicollas Mocelin Sdroievski. Instance-wise hardness versus randomness tradeoffs for Arthur-Merlin protocols. In *Computational Complexity Conference (CCC)*, 17:1–17:36, 2023.

[vMM23b] 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)*, 29:1–29:22, 2023.

[vMP19] Dieter van Melkebeek and Gautam Prakriya. Derandomizing isolation in space-bounded settings. *SIAM Journal on Computing*, 48(3):979–1021, 2019.

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