UNIVERSITY OF CALIFORNIA SAN DIEGO

Connections Between Complexity Lower Bounds and Meta-Computational Upper Bounds

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Computer Science

by

Marco Leandro Carmosino

Committee in charge:

Professor Russell Impagliazzo, Chair
Professor Samuel Buss
Professor Shachar Lovett
Professor Ramamohan Paturi
Professor Gila Sher

2019

The Dissertation of Marco Leandro Carmosino is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California San Diego

2019

EPIGRAPH

Reality is that which, when you stop believing in it, doesn't go away.

Phillip K. Dick

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

ACKNOWLEDGEMENTS

I would like to thank my advisor Russell Impagliazzo for his support and guidance. I would also like to acknowledge my other co-authors Valentine Kabanets, Antonina Kolokolova, and Manuel Sabin. Our collaborations made this dissertation possible.

Chapter 3, in part, is based on the material as it appears in "Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Learning algorithms from natural proofs. In Ran Raz, editor, *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, volume 50 of *LIPIcs*, pages 10:1–10:24. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016". The dissertation author was the primary investigator and author of this paper.

Chapter 4, in part, is based on the material as it appears in "Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Agnostic learning from tolerant natural proofs. In Klaus Jansen, José D. P. Rolim, David Williamson, and Santosh Srinivas Vempala, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, volume 81 of *LIPIcs*, pages 35:1–35:19. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017". The dissertation author was the primary investigator and author of this paper.

Chapter 5, in part, is based on the material as it appears in "Marco L. Carmosino, Russell Impagliazzo, and Manuel Sabin. Fine-grained derandomization: From problem-centric to resource-centric complexity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 27:1–27:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018". The dissertation author was the primary investigator and author of this paper.

Chapter 6, in part, is based on the material as it appears in "Marco Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Tighter connections between derandomization and circuit lower bounds. In Naveen Garg, Klaus Jansen, Anup Rao, and

José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA*, volume 40 of *LIPIcs*, pages 645–658. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015". The dissertation author was the primary investigator and author of this paper.

VITA

2011 Bachelor of Arts, Hampshire College

2013 Master of Science, University of Massachusetts Amherst

2019 Doctor of Philosophy, University of California, San Diego

ABSTRACT OF THE DISSERTATION

Connections Between Complexity Lower Bounds and Meta-Computational Upper Bounds

by

Marco Leandro Carmosino

Doctor of Philosophy in Computer Science

University of California San Diego, 2019

Professor Russell Impagliazzo, Chair

This dissertation presents several results at the intersection of complexity theory and algorithm design. Complexity theory aims to lower-bound the amount of computational resources (such as time and space) required to solve interesting problems. Algorithm design aims to upper-bound the amount of computational resources required to solve interesting problems. These pursuits appear opposed. However, some algorithm design and complexity lower bound problems are inextricably connected.

This dissertation explores several such connections. From "natural" proofs of circuit-size lower bounds, we create learning algorithms. From the exact hardness of problems in polynomial time, we create algorithms of estimating the acceptance probability of circuits. Finally, from

algorithms for testing the identity of arithmetic circuits over finite fields, we create arithmetic circuit-complexity lower bounds.

Chapter 1

Introduction

Complexity theorists seek the limits of efficient computation. They try to *lower bound* the amount of computational resources (eg. time, storage, random bits, communication, observations) required to solve problems. Algorithm designers try to develop efficient algorithms for problems, giving *upper bounds* on the computational resources required for these tasks. These goals appear opposed. Yet a complexity lower bound also contains information about efficient computation; it must somehow identify a weakness of *any* efficient computation.

This thesis studies ongoing attempts to unify algorithm design and complexity theory. There are entire classes of problems whose solutions rely on understanding and exploiting the *simplicity* of efficient computation — the types of weaknesses identified by complexity lower bounds. Towards characterizing such problems, we seek formal answers to the fundamental questions:

How and when can an algorithm be extracted from a complexity lower bound?
For what algorithmic tasks are complexity lower bounds *necessary*?

The natural starting point is problems whose inputs may be computations themselves; we call such problems *meta-computational*. Automatic program synthesis, circuit property testing, and derandomization are just a few examples. Any task where the goal is to transform, analyze, or construct another computational object is meta-computational. We shall see that algorithm design for these tasks is inextricably linked to progress on complexity lower bounds.

To define some meta-computational problems formally, first fix a *complexity class* Λ:

a set of functions that can be computed within certain resource constraints. We often define complexity classes by imposing structure and size restrictions on *circuits* — devices composed of small "gates," each computing a simple function like AND, OR, NOT. Below, we introduce some fundamental meta-computational problems, where Λ is a variable complexity class. Notice that the way a Λ-function is presented can vary; it could be a circuit, a "black box," or an explicit list of all function values (a truth table). The unifying feature is that, however an input function is presented, it is guaranteed to come from the class Λ.

Meta-Computational Problems:

- *Circuit Acceptance Probability Problem* (Λ-CAPP): given a Boolean circuit C from class Λ, estimate the acceptance probability of C to within small additive error; the quantity $\Pr_x[C(x) = 1] \pm 1/10$.

- *Circuit Satisfiability* (Λ-Circuit-SAT): given a Boolean circuit C from class Λ, decide if there exists an input to C that makes it *accept* (ie, print '1')

- *Learning* (Λ-LEARN): given only the ability to query a Boolean function f from Λ, output a small circuit approximating f.

- *Minimum Circuit Size Problem*, (Λ-MCSP): given the complete truth table of a Boolean function f and a number s, is there a Λ-circuit of size less than s that computes f?

Intuitively, there are connections between meta-computation and lower bounds because both endeavors require a deep understanding of *all* Λ-functions: either to automatically analyze them, or to prove that some "hard" function is not in Λ. The relationships established by this thesis include:

- Structured *proofs* of complexity lower bounds against Λ can be transformed into Λ-LEARN *algorithms*. (Chapters 3 and 4)

- Popular conjectures of "fine-grained" lower bounds for concrete problems imply *very efficient* solutions to Λ-CAPP. (Chapter 5)

- It is impossible to give a non-trivial algorithm for Λ-CAPP without proving arithmetic circuit lower bounds over finite fields. (Chapter 6)

Before sketching these results in more detail, we give a compressed and biased overview of complexity theory.

1.1 Historical Context

Computation

Turing Machines (TMs) were introduced in 1936, as the foundation of Alan Turing's proof that there is no algorithm for deciding if a sentence of first-order logic is universally valid [Tur36]. This is a "computability" lower bound. Turing Machines provided a framework for defining general-purpose computation and constructing programmable systems and algorithms. Much early work in computability theory resulted in the understanding, summarized by the "Church-Turing Thesis", that every reasonable definition of "computation" coincides. Thus, the notion of an algorithm — a purely mechanical procedure for information processing — is robust. But the foundations of computer science could not distinguish between problems solvable by reasonable computers in "a minute" and "the expected lifetime of our universe."

Computational Complexity

Juris Hartmanis and R. E. Stearns handled this issue, by inventing Computational Complexity [HS65]. Suppose we have created a Turing Machine M for solving a particular problem L. By "observing" the behavior of a M while it operates, we can upper-bound the amount of resources it will use (such as tape cells or time steps) to solve L. After this work, computational models proliferated by defining different types of Turing Machines and counting different types of resources. These resources are used to define *complexity classes* of problems solvable by algorithms within given amounts of resources. Importantly, Hartmanis and Stearns were able to

show (using diagonalization) that "more time means more power." These results are some of the first examples of complexity lower bounds.

Theorem 1 (Informal Example of a Complexity Lower Bound [HS65])**.** *There exists a decision problem L which* ***cannot*** *be solved on a TM using $t(n)$ steps, but* ***can*** *be solved on a TM using $t(n)^2$ steps on binary inputs of length n.*

Lower-bounds based on diagonalization are often called "Hierarchy Theorems." These theorems have two limitations:

1. The hard languages that witness separation between (for example) $t(n)$ steps and $t(n)^2$ steps are entirely artificial. We would like complexity lower-bounds for natural problems.

2. The hierarchy theorems can only treat a single resource. That is, they compare "more time" to "less time" and "more space" to "less space," but they do not compare time to space.

Tractable Problems

Shortly after the introduction of machine-based complexity theory, Alan Cobham and Jack Edmonds separately identified the class of problems decidable in a polynomial number of steps as "tractable," and this definition has served as the first step in understanding the complexity of a problem since then [Cob65, Edm65]. Formally, a problem is in P (for "polynomial time") if there exists some k such that input instances of n-bit size can be decided in at most n^k steps on a deterministic Turing Machine. Though it abstracts many details, this identification of P with feasible problems at least roughly classifies problems according to their time complexity.

Intractable(?) Problems

The central mystery of computational complexity was independently identified by Cook and Levin [Coo71, Tra84]. Suppose that we can *efficiently recognize* the solutions to a search problem S. That is, given a problem instance and a candidate solution, we can decide if the solution is valid in P. We call the class of such problems NP. Though many NP problems *appear*

to require brute-force search over the solutions space and thus exponential (not polynomial) time in the size of the input instance, we cannot (yet?) prove this.

Open Problem 2. *Is* P *equal to* NP *?*

Another way to define NP is as the set of languages that are solvable using a *nondeterministic* TM — a TM that may "guess" what to do next. Though we can prove hierarchy theorems, separately, for nondeterministic time and deterministic time, we cannot *relate* these resources to each other and get a separation between P and NP

Randomness as a Resource

Early work studied many variants of the Turing Machine model. It was shown that we can translate algorithms between one-tape, multi-tape, and random-access models with very little simulation overhead. It became clear that many "mechanical" changes to the underlying notion of a Turing Machine simply *do not matter*, up to small quantitative factors. What about more radical changes?

Suppose we allow the machine to flip coins. Consider probabilistic Turing Machines — machines that have access to a separate input tape of perfectly random bits [Gil77]. Many problems have more efficient or simpler algorithms if the TM is allowed to make random choices. The natural complexity class capturing efficient randomized computation is called BPP, for "bounded-error probabilistic polynomial time." We would like to reap the performance and efficiency benefits of randomized algorithms using deterministic computational devices. But, currently, we do not know how to efficiently "de-randomize" algorithms.

Open Problem 3. *Is* P *equal to* BPP *?*

The limitations of current knowledge are far more severe than just an inability to resolve "tractable" derandomization; we do not even understand *non-trivial* derandomization. Consider the trivial derandomization of BPP: for each possible random string r, which must be of polynomial length — because otherwise, the algorithm would not have time to read it — simulate

the BPP algorithm on given input x and random string r. Then, count the number of ACCEPT outcomes. Accept if the fraction of ACCEPT outcomes is large enough. This trivial procedure takes exponential time, but it is the *best known* generic derandomization of BPP. Denote by EXP the complexity class of exponential-time computations on a deterministic TM. The foregoing discussion raises the following:

Open Problem 4. *Is* EXP *equal to* BPP *?*

Concrete Complexity

So far, our historical vignettes have all concerned Turing Machines. Turing Machines are a *uniform* model of computation, in the sense that a single TM encodes an algorithm that processes inputs of any size. Thus, TMs model the complexity of software running on a general-purpose computer. This neglects the complexity of the *underlying hardware.* To actually build a computer, we must synthesize circuits that operate on fixed-length "words" and perform operations like addition, multiplication, multiplex, etc.

A theoretical account of such circuits was introduced by Shannon in 1938. Briefly: we fix a set of Boolean functions, call them "gates," and connect their inputs and outputs by wires. We can measure the complexity of circuits by counting the number of gates (size) or the length of the longest path from input to output (depth) [Sha38].

In contrast to machine-based complexity, circuit complexity is *non-uniform*. A Turing Machine processes inputs of arbitrary size, a circuit processes inputs of fixed size. So to speak about solving a decision problem with circuits, we must say that there is a *family* of circuits, one for each input length. As with machine-based complexity, we identify "tractability" with polynomial bounds on resources. The class of polynomial-sized circuits is called P/poly, and considered the "benchmark" tractable circuits class.

By the Deterministic Time Hierarchy theorems alluded to above, we know:

Theorem 5 (Uniform Separation)**.** $\mathsf{P} \neq \mathsf{EXP}$

What about P/poly, the circuit analog of P? In this case, diagonalization techniques fail, and we are faced with:

Open Problem 6. *Does* P/poly *contain* EXP *?*

Hardness Implies Randomness

We conjecture that reasonable-seeming *circuit lower bounds* are true — that, is that the answer to Open Problem 6 above is **no**. Intuitively, it "should" be the case that problems requiring exponential time to compute on a deterministic TM do not have polynomial-sized circuits; how could allowing a different device at each input length so radically alter the complexity of *every* problem solved by a deterministic TM? If $\mathsf{EXP} = \mathsf{P}/\mathsf{poly}$, it would mean that the execution of *every* deterministic computation can be radically "compressed" by non-uniform devices. But we cannot yet prove strong enough circuit lower bounds to refute "$\mathsf{EXP} = \mathsf{P}/\mathsf{poly}$", and the role that circuit lower bounds should play in the structural complexity theory of Turing Machine variants is not clear at first glance.

A breakthrough result of Nisan and Wigderson showed that if plausible circuit lower bounds such as "$\mathsf{EXP} \neq \mathsf{P}/\mathsf{poly}$" are true, then any algorithm in BPP can be non-trivially simulated by a deterministic algorithm [NW94]. Intuitively, they use the circuit lower bounds to generate patters that are so complex that they *appear* random to algorithms with bounded resources. This insight sparked a rich line of work on hardness vs randomness *trade-offs*, of which the following summative result is representative:

Theorem 7 (High-End Hardness to Randomness, [IW97])**.** *If reasonable circuit lower bounds hold against* P/poly *then* $\mathsf{P} = \mathsf{BPP}$

This result lends even more motivation to proving circuit lower bounds. If we had circuit lower bounds, not only would we understand concrete complexity better, but we could efficiently derandomize *any* randomized algorithm and answer Open Question 3 in the most useful way possible; any efficient randomized algorithm could be simulated by an efficient deterministic algorithm.

Proving Circuit Lower Bounds Seems Hard

Rapid progress was made on circuit lower bounds in the 80's. Researchers proved lower bounds against increasingly powerful sub-classes of $\mathsf{P/poly}$ [FSS84, Raz87, Smo87]. But progress stalled. An explanation for this stall was developed by Razborov and Rudich. They showed that if existing circuit-lower bound techniques could be extended to work against $\mathsf{P/poly}$, then cryptography is impossible [RR97]. Because researchers generally conjecture that cryptography works, this means that attempts to prove circuit lower bounds against $\mathsf{P/poly}$ using the (very broad) class of arguments identified by Razborov and Rudich are doomed to failure.

1.2 Chapter Overview

We can now sketch in more detail the contents and contributions of each chapter.

Distinguishers Imply Predictors.

This chapter describes the fundamental results from cryptography and pseudorandomness that we use to study meta-computation and complexity lower bounds. Briefly, the ability to distinguish structure from random noise entails the ability to learn concepts from a teacher. Every subsequent chapter relies on this primitive. So, we present it first and in some detail. This "distinguisher to predictor" construction is implicit in [NW94], the breakthrough result connecting circuit lower bounds and derandomization.

Natural Circuit Lower Bounds Imply Learning Algorithms.

Most known complexity lower bounds fit the *Natural Proofs* framework of [RR97]. Natural Proofs against a complexity class Λ implicitly contain an algorithm that distinguishes between truth-tables of Λ-circuits and random strings with high probability; this can be viewed as solving an approximate version of the Λ-MCSP problem described above. Razborov and Rudich showed (under widely-believed cryptographic assumptions) that Natural Proofs cannot separate P from NP. Thus, the ability of the Natural Proofs framework to capture most known complexity lower bounds is generally cited as a barrier to separating P from NP.

My joint work [CIKK16] covered in Chapter 3 of this thesis shows that the Natural Proofs framework is also a powerful tool for meta-computational algorithm design. We give a generic construction of Λ-LEARN algorithms from Natural Proofs against a class Λ, for any Λ satisfying a certain mild technical condition — Λ must locally compute the design functions from [NW94]. Our construction immediately yielded the first non-trivial (quasi-polynomial time) learning algorithms for $\mathsf{AC}^0[p]$, the class of functions computed by constant-depth circuits of polynomial size with AND, OR, NOT and counting modulo p gates (where p is a prime). Obtaining any kind of learning algorithm for $\mathsf{AC}^0[p]$ was open for 23 years. Previous work gave a learning algorithm for AC^0 (the same class without counting modulo p) in 1993 [LMN93]. Our main construction has been reused to make progress on questions about learning, complexity, and cryptography [OS17, IKV18, OS18, Hir18].

While prior work has also developed algorithms from complexity lower bounds, ours is a rare example of a generic construction. The AC^0-SAT, AC^0-LEARN, and AC^0-COMPRESS algorithms (and indeed, *all* known algorithms for Boolean meta-computation) were developed by adapting Natural proof techniques originally intended for lower bounds against AC^0 [IMP12, LMN93, CKK$^+$15]. Those previous results manually inspect the lower bound proof to uncover a concrete weakness of the target complexity class, and then exploit it for algorithm design. Our learning algorithm only needs two pieces of information about Λ: a Natural Proof, which is treated as a black box, and that Λ satisfies the "local design function" technical condition.

This genericity in our work suggests a structural relationship between proving circuit lower bounds and constructing algorithms. In future work, I seek to develop a new algorithm design paradigm tailored to meta-computation, by constructing frameworks for complexity lower bounds that are meant to be translated into algorithms for meta-computational problems.

Tolerant Natural Circuit Lower Bounds Imply Agnostic Learning Algorithms.

This chapter makes some progress towards generalizing the relationship between Natural circuit lower bounds and learning algorithms. In subsequent work with the same co-authors, I

showed that many Natural Proofs can actually distinguish between random strings and strings merely close to the truth-table of a Λ-function [CIKK17]. This enables our learning algorithms (once suitably modified) to tolerate adversarially corrupted answers to queries, obtaining the first *agnostic* learning algorithms for $\mathsf{AC}^0[p]$. This is a different and more difficult learning model. Successful construction of a proof-to-algorithm transformation for this model as well suggests that we should continue to pursue this direction.

Popular fine-grained hardness conjectures imply derandomization.

Fine-grained complexity refines the notion of a tractable problem introduced above. As problem sizes scale up, the difference between a $O(n^2)$-time algorithm and a $O(n)$-time algorithm becomes more pronounced. Fine-grained complexity examines the internal structure of BPP, distinguishing between randomized time $O(n^k)$ and $O(n^{k+1})$ for every k. We conjecture that some problems form parameterized hierarchies, where for each k the k-parameter version of the problem can be solved in n^k but **not** $n^{k-\varepsilon}$ time.

Usually, hardness implies derandomization via some version of the hardness to randomness tradeoff [NW94]. But the key problems studied in fine-grained complexity are generally conjectured to be hard for *randomized algorithms,* instead of non-uniform circuits. Thus standard hardness-to-randomness tradeoffs cannot conclude derandomization from these conjectures; the tradeoffs requires hardness against non-uniform circuits.

However, uniform hardness is known to imply an *approximate* simulation of BPP [IW01, TV07]. So we should be able to conclude some kind of approximate simulation of BPP from the popular conjectures of fine-grained complexity. This does not quite work either — the derandomizations from a uniform hypothesis are only *low-end* hardness-to-randomness results. They convert weak hardness ($\mathsf{EXP} \neq \mathsf{BPP}$) into slow (sub-exponential time) deterministic approximations of BPP. A *high end* uniform derandomization — converting strong hardness assumptions into fast deterministic approximations of BPP — is open.

In Chapter 5, we show that the hardness assumptions about key problems in fine-grained

complexity imply fast deterministic approximations of BPP. To accomplish this, we bring the seed-extending pseudorandom generators of [KvMS12] into the uniform derandomization framework mentioned above, and take advantage of recent worst-case to average-case reductions developed for these key problems by [BRSV17] and the "nice" structural properties of key problems studied in fine-grained complexity. Our work represents progress towards truly generic high-end uniform derandomization — our techniques work only when hardness is assumed for particularly "nice" problems.

Some Circuit Lower Bounds are *equivalent* to Derandomization.

In Chapter 6, we tighten the connections between circuit lower bounds and derandomization for each of the following three types of derandomization:

- general derandomization of promise-BPP (connected to Boolean circuits),
- derandomization of Polynomial Identity Testing (PIT) over fixed finite fields (connected to arithmetic circuit lower bounds over the same field), and
- derandomization of PIT over the integers (connected to arithmetic circuit lower bounds over the integers).

We show how to make these connections *uniform equivalences*, although at the expense of using somewhat less common versions of complexity classes and for a less studied notion of inclusion. Our main results are as follows:

1. We give the first proof that a non-trivial (nondeterministic subexponential-time) algorithm for PIT over a *fixed finite field* yields arithmetic circuit lower bounds.
2. We get a similar result for the case of PIT over the integers, strengthening a result of Jansen and Santhanam [JS12] (by removing the need for advice).
3. We derive a Boolean circuit lower bound for NEXP $\cap$ coNEXP from the assumption of sufficiently strong non-deterministic derandomization of promise-BPP (without advice), as

well as from the assumed existence of an NP-computable non-empty property of Boolean functions useful for proving superpolynomial circuit lower bounds (in the sense of natural proofs of [RR97]); this strengthens the related results of [IKW02].

4. Finally, we turn all of these implications into equivalences for appropriately defined promise classes and for a notion of robust inclusion/separation (inspired by [FS11]) that lies between the classical "almost everywhere" and "infinitely often" notions.

1.3 Common Elements

In each result of this thesis, the Nisan-Wigderson pseudorandom generator (NW-PRG) plays a key role. In Chapters 3 and 4, we treat the analysis of this PRG as a learning algorithm. In Chapter 5, we use the NW-PRG as a PRG to obtain uniform derandomizations, again taking advantage of the algorithmic nature of the analysis of the generator. In Chapter 6, we use a version of the NW-PRG tailored for arithmetic circuits (due to [KI04]) to establish the equivalences sketched above.

What is highlighted here is the enormous flexibility obtained by interpreting the proof that an algorithm is correct as, itself, an algorithm. Thus, we begin the technical content of this thesis in the next chapter by explaining the ideas behind the NW-PRG, and showing how they are amenable to being recast as an algorithm.

Chapter 2

The Nisan-Wigderson Distinguisher to Predictor Transformation

2.1 Introduction

"Recognition" is the foundational task of modern learning theory. Intuitively, to "recognize" a concept means the ability to discriminate between positive and negative examples of the concept. For example, I can "recognize" the difference between pictures of kittens and pictures of non-kittens — because the pictures of non-kittens are far less pleasing. This facility is part of my knowlege about the concept of "kitten." We refer to this very concrete type of recognition as *prediction*.

A more subtle type of recognition is detecting the **presence** or **absence** of *patterns in general.* I can also perform this task. When shown a succession of images, some of which were generated by flipping a coin at each pixel and some of which were taken with a camera, I am (generally) able to "recognize" the structures present in a camera image and correctly label those images as non-random. We will refer to this more abstract type of recognition as *distinguishing*.

Early work in cryptography observed a beautiful relationship between distinuishing and predicting [Yao82]. Very roughly, the ability to *distinguish* patterns in general from random noise can be used to *predict* concrete patterns supplied by a teacher. This thesis exploits the connection between distinguishing and predicting to obtain new results in both computational complexity theory and learning theory.

In this chapter, we describe the Nisan-Wigderson (NW) generator construction [NW94]. This is the starting point for all of our distinguisher to predictor transformations. Informally, this construction takes as input the truth table of a Boolean function f, and outputs a circuit for the new function G_f mapping "short" input strings to "long" output strings. The function G_f is intended to be a *pseudo-random generator (PRG)* in the sense that no "small" Boolean circuit can "distinguish" the uniform distribution from the distribution of G_f's outputs (on uniformly random inputs to G_f). A circuit that can distinguish these two distribution is said to break the generator, and is called a *distinguisher*. Nisan and Wigderson [NW94] prove that if the initial function f has "high" circuit complexity, then the function G_f is indeed a PRG. Moreover, their proof is constructive in the sense that there is an efficient *reconstruction* algorithm that, given a distinguisher for G_f and oracle access to f, outputs a "small" Boolean circuit that approximately computes f. Thus, the NW construction has dual applications, both demonstrated in this thesis: learning (Chapters 3 and 4) and derandomization (Chapters 5 and 6). We define the generator formally below, and spend the rest of this chapter describing the associated reconstruction algorithm.

Our presentation is elementary and follows [AB09], while introducing notation that will make our later applications of the construction clear. We begin by giving formal definitions of what it means to "predict" and "distinguish" under resource constraints.

2.2 Formal Preliminaries

Some notation and abbreviations about circuits will be helpful later. All definitions "native" to the construction will be introduced inline. Let μ be some distribution and C be a Boolean circuit. We write $C(\mu)$ to abbreviate $\Pr_{x\sim\mu}[C(x) = 1]$. We abbreviate "$C(x) = 1$" by $C(x)$. A *transducer* is a multi-output Boolean function. When referring to variable complexity classes, we often use Λ to mean a "weak" class and Γ to mean a "strong" class.

2.2.1 Circuits and Circuit Construction Tasks

For a circuit class Λ and a set of size functions $\mathscr{S}$, we denote by $\Lambda[\mathscr{S}]$ the set of $\mathscr{S}$-size n-input circuits of type Λ. When no size class is explicitly given, $\mathscr{S}$ is assumed to be $\mathsf{poly}(n)$. Below we formalize computational distinguishability and approximate prediction using circuits. Then we introduce notation for algorithmic transformations of circuits.

Definition 2.2.1 (Circuits (Approximately) Computing f)**.** Let $f : \{0,1\}^n \to \{0,1\}$ be some Boolean function, and let $\varepsilon : \mathbb{N} \to [0,1]$ be an approximation bound. Then $\mathscr{C}^f$ denotes the set of circuits that compute the function f on all n-bit inputs, and $\hat{\mathscr{C}}^f_\varepsilon$ the set of all circuits that compute f on all but an ε fraction of inputs.

Definition 8 (Distinguishers)**.** Let $L\colon \mathbb{N} \to \mathbb{N}$ be a stretch function, let $0 < \gamma < 1$ be a gap bound, and let $\mathscr{G} = \{g_m\colon \{0,1\}^m \to \{0,1\}^{L(m)}\}$ be a sequence of transducers. A *distinguisher for $\mathscr{G}$ with distinguishing gap γ*, or a *γ-distinguisher against $\mathscr{G}$,* is a circuit from the set

$$\mathsf{DIS}(\mathscr{G},\gamma) := \text{ circuits } D \text{ on } L(m) \text{ inputs such that } \left| \Pr_{z\in\{0,1\}^m}[D(g_m(z))] - \Pr_{y\in\{0,1\}^{L(m)}}[D(y)] \right| > \gamma$$

Definition 2.2.2 (Circuit Builder Declarations — adapted from [IW01])**.** Let A and B be indexed sets of circuits. A *$T(n)$-construction of B from A* is a probabilistic machine $\mathscr{M}(n,\alpha,A_n)$ which outputs a member of B_n with probability at least $1-\alpha$ in time $T(n)$, where the size of B_n is $\mathsf{poly}(|A_n|)$. We declare that such a machine exists by writing: $A \longmapsto B$ in $\mathsf{TIME}[T(n)]$. Read this notation as "from A we can construct B in time $T(n)$." To assert the existence of a *$T(n)$-construction of B from A, with oracle $\mathscr{O}$*, where the machine $\mathscr{M}$ is equipped with an oracle for the language $\mathscr{O}$ but otherwise is as above, we write: $A \longmapsto B$ in $\mathsf{TIME}^{\mathscr{O}}[T(n)]$.

2.3 Nisan-Wigderson Construction

We now define the NW generator formally. First we require *designs*, set systems with some nice properties.

Definition 9 (Nisan-Wigderson (NW) Design)**.** For parameters $n, m, L \in \mathbb{N}$, a sequence of sets $S_1, \ldots, S_L \subseteq [m]$ is called an *NW design* if the sets satisfy:

- **Fixed Size:** $|S_i| = n$, for all $1 \leq i \leq L$, and
- **Low Overlap:** $|S_i \cap S_j| \leq \log L$, for all $1 \leq i \neq j \leq L$.

Nisan-Wigderson designs exist and can be efficiently constructed for any n, $m = O(n^2)$, and $L < 2^n$ [NW94], even in $\mathsf{AC}^0[\oplus]$. We give a such an efficient construction of NW-Designs in Section 3.3.1, as this is the key technical component of the learning algorithm for $\mathsf{AC}^0[p]$ presented in Chapter 3 of this thesis. The NW generator will use designs to evaluate f on many low-overlap substrings of a "seed" string, building up a much longer output string.

Definition 10 (NW Generator)**.** Let $f\colon \{0,1\}^n \to \{0,1\}$. For $m = n^2$ and a stretch function $L(m)\colon \mathbb{N} \to \mathbb{N}$, where $L(m) < 2^n$, let $S_1, \ldots, S_L \subseteq [m]$ be a NW design. Define the NW generator $\mathsf{NW}(f) = G^f\colon \{0,1\}^m \to \{0,1\}^{L(m)}$ mapping seeds z to $L(m)$-length strings as:

$$G^f(z) = f(z\restriction_{S_1}) f(z\restriction_{S_2}) \ldots f(z\restriction_{S_{L(m)}}), \tag{2.1}$$

where $z\restriction_S$ denotes the $|S|$-length bit-string obtained by restricting z to the bit positions indexed by the set S.

Theorem 2.3.1 (Uniform NW Reconstruction [NW94, IW01])**.** *There is a uniform randomized algorithm that, given as input a distinguisher circuit D for the NW Generator defined using f and an oracle for f, will print a circuit that approximately computes f. Formally:*

$$\mathsf{DIS}(G_f, 1/5) \longmapsto \hat{\mathscr{C}}^f_{1/2-1/L} \ \textit{in} \ \mathsf{TIME}^f[\mathsf{poly}(L)]$$

Usually, this reconstruction theorem is used for hardnes vs randomness trade-offs by taking the contrapositive: if f is hard to approximate using small circuits, then it must be the case that no small circuit can distinguish between truly random bits and the output of G_f. For

this standard use of the NW Reconstruction theorem, it does not matter that reconstruction is algorithmic because circuits are a *non-uniform* model of computation.

All through this thesis, however, we emphasize the benefits of understanding this reconstruction procedure as an *algorithm*. So we use the reminder of this chapter to sketch the key algorithmic ideas of NW Reconstruction.

2.4 Non-uniform NW Reconstruction

First we describe the *non-uniform* map from a distinguisher against G^f to an approximator for f. This proof is simple and contains the main ideas used by the reconstruction *algorithm*, so we present it first. Two key insights of [NW94] drive the proof:

1. **Distinguishing Hybrids:** The ability to noticably distinguish fully random sequences from the output of G^f implies the ability to distinguish between *partially* random sequences mixed with the output of G^f. This is proved by a *hybrid argument*.

2. **Constrained Evaluation:** The output of G^f is a sequence of mostly-independent evaluations of f, so there are *few degrees of freedom left* in the output space of the generator once a seed and a design index are fixed.

2.4.1 Hybrid Argument

First, we formalize the notion of mixing generator outputs with true randomness.

Definition 11 (Generic Hybrids)**.** Let $G : \{0,1\}^m \to \{0,1\}^{L(m)}$ be a transducer. Define a family of distributions $\mathscr{W}_0, \ldots, \mathscr{W}_{L(m)}$, each over binary strings $w_1 w_2 \ldots w_{L(m)} \in \{0,1\}^{L(m)}$ by the following process: $\mathscr{W}_i$-SAMP :

1. Sample a seed $z \in \{0,1\}^m$ uniformly at random

2. For each $j \leq i$, $w_j \leftarrow G(z)_j$ // place the jth bit of $G(z)$ in w_j

3. For each $j > i$, $w_j \leftarrow \mathscr{U}$ // place a bit chosen uniformly at random in w_j

Think of the hybrid index i as "number of bits used from the generator." First consider step (2) applied to $\mathscr{W}_0$: $\forall j \leq i$, $w_j \leftarrow G(z)_j$. Here $i = 0$, and there are no indices less than or equal to zero in w. So every bit of $\mathscr{W}_0$ is sampled uniformly at random in step (3). Therefore:

$$\mathscr{W}_0 \equiv U_{L(m)} \equiv \text{ Purely random bits}$$

On the other hand, consider step (2) applied to $\mathscr{W}_{L(m)}$: $\forall j \leq i$, $w_j \leftarrow G(z)_j$. Here $i = \ell$, which is the last index of w, so after step (2) every bit is filled with generator output. Step (3) is a no-op, because there are no indicies larger than $L(m)$ in w. Therefore:

$$\mathscr{W}_{L(m)} \equiv G(\mathscr{U}_m) \equiv \text{ Full transducer output}$$

We now show that if D is a circuit that can distinguish between these extremal points (generator output and $\mathscr{U}$ this means that D has some expected advantage at distinguishing between randomly chosen adjacent elements of $\mathscr{W}$.

Lemma 12 (Adjacent Hybrids are Distinguishable)**.** *If $D \in \mathsf{DIS}(G, \gamma)$ and G is a PRG with stretch L, then we can expect a non-trivial distinguishing gap for randomly chosen adjacent pairs of hybrids. Formally:*

$$\mathop{\mathbb{E}}_{i \in [L]} [D(\mathscr{W}_i) - D(\mathscr{W}_{i-1})] \geq \gamma / L$$

Proof (by hybrid argument). For each $i \in [L]$, let $p_i = \Pr[D(\mathscr{W}_i)]$. By possibly negating the top gate of D', we can obtain D such that:

$$\Pr[D(G(\mathscr{U}_m))] - \Pr[D(\mathscr{U}_{L(m)})] \geq \gamma \tag{2.2}$$

Rewrite the distinguishing equation about D as "$p_L - p_0 \geq \gamma$". Then expand this distinguishing gap into a summation over intermediate i. All terms in the expansion except p_L and p_0

appear both positively and negatively, telescoping the sum and giving the first equality below.

$$\begin{aligned} p_L - p_0 &= (p_L - p_{L-1}) + (p_{L-1} - p_{L-2}) + \cdots + (p_1 - p_0) \\ &= \sum_{i \in [L]} (p_i - p_{i-1}) \\ &\geq \gamma \end{aligned}$$

Multiplying through by $1/L$ and rewriting the results using $\mathbb{E}$ completes the proof:

$$\begin{aligned} (1/L) \sum_{i \in [L]} (p_i - p_{i-1}) &\geq \frac{\gamma}{L} \\ \mathbb{E}_{i \in [L]}[p_i - p_{i-1}] &\geq \frac{\gamma}{L} \end{aligned}$$

□

Note that there is a loss in the gap at each hybrid pair, proportional to the PRG stretch/number of hybrids. This loss seems unavoidable by a generic hybrid argument [FSUV13].

2.4.2 Predictor Strategy

Given the ability to distinguish between G^f hybrid pairs demonstrated above, we can gain advantage in predicting f by believing the distinguisher circuit D on carefully-crafted inputs.

Intuitively, this strategy (Algorithm 1) relies on D's distinguishing gap between hybrids. If the guessed bit g agrees with $f(x) = f(z|_{S_i})$, then w looks more like a sample from $\mathscr{W}_i$ because $f(z|_{S_i})$ is the ith bit of generator output. If the guessed bit disagrees with $f(x)$, then w looks more like a sample from $\mathscr{W}_{i-1}$ because g was selected at random. We know that D is biased (in expectation) towards samples from $\mathscr{W}_i$ by Lemma 12. So if $D(w) = 1$, this is evidence that the

1. Guess $f(x)$ and call this bit g
2. Randomly select $i \in [L(m)]$, to pick out a hybrid pair
3. Sample $w \sim \mathscr{W}_i$ consistent with $g = w_i$
4. Run $D(w)$ and trust it, answering accordingly:

$$\mathsf{Out}(x) = \begin{cases} g & \text{if } D(w) = 1 \\ & \text{// } w \text{ is a } \mathscr{W}_i \text{ sample} \implies \text{we guessed } f(x) \text{ right} \\ \neg g & \text{if } D(w) = 0 \\ & \text{// } w \text{ is a } \mathscr{W}_{i-1} \text{ sample} \implies \text{we guessed } f(x) \text{ wrong} \end{cases}$$

Algorithm 1. Informal Randomized Predictor Strategy

guess g is correct, so we can obtain some advantage in predicting f by printing g. Similarly, if $D(w) = 0$, this is evidence that the guess disagrees with $f(x)$, so we should output $\neg g$.

The strategy samples from $\mathscr{W}_i$ in step 3. This seems circular in a procedure attempting to predict f, as sampling from $\mathscr{W}_i$ certainly requires access to at least i values of f. The designs will let us "sample" from $\mathscr{W}_i$ using only a reasonably-sized lookup table[1] of f-values, instead of access to arbitrary evaluations of f. This is one of the key innovations of the NW construction. So we can begin by assuming an oracle for these $\mathscr{W}$-samples, and prove the strategy outlined above does indeed work.

2.4.3 Randomized Next-Bit Predictor

Let $\vec{y}$ denote $y_1, \ldots, y_{i-1}$, the first $(i-1)$ bits of $G^f(z)$ for $z \sim \mathscr{U}_m$. In this section we'll show that a distinguisher can give us expected advantage in computing the next bit of $G^f(z)$. Essentially, we analyse the correctness of the strategy above without considering the complexity of sampling from $\mathscr{W}_i$, by just *handing* our predictor the part of a $\mathscr{W}_{i-1}$ sample that depends on f-values. The idea and construction is due to Yao [Yao82]. Our presentation here follows [AB09],

[1] The final construction will not literally sample from the ith hybrid. It will store a "good enough" set of samples from the ith hybrid and use those in the predictor.

while giving additional details which help to describe the uniform reconstruction algorithm. Note that this next-bit predictor works for any transducer G, but the part of the argument where we remove dependence on $\mathscr{W}$ samples will be specific to the NW Generator.

1. $w \leftarrow 0^L$ // initialize a string of length L
2. $w \leftarrow \vec{y}$ // fill w_0 to w_{i-1} with PRG output "so far"
3. For each j from i upto L:
   $w_j \leftarrow \mathscr{U}$ // fill remainder of w with randomness – implicitly guessing $f(x)$
4. $$\mathsf{Output} \leftarrow \begin{cases} w_i & \text{if } D(w) = 1 \\ \neg w_i & \text{if } D(w) = 0 \end{cases}$$

Algorithm 2. Next-Bit Predictior, $\text{NB}_i(\vec{y})$

Lemma 13 (Distinguisher to Randomized Next-Bit Predictor)**.** *When constructed from $D \in \mathsf{DIS}(G, \gamma)$ the* NB *circuit has non-negligable advantage in predicting the next bit of G.*

$$\mathbb{E}_{i \in \ell}\left[\Pr_{r,z}[\text{NB}(\vec{y}, r) = y_i]\right] \geq (1/2) + \gamma/L$$

Proof. Denote the event "$\text{NB}(\vec{y}) = y_i$" by PC. Because w_i is a random bit, we can think of it as a guess of y_i. There are two events that imply NB is correct:

1. The "guessed bit" is correct and D accepts w. So, $D(w) = 1$, and $w_i = y_i$, which we denote cg
2. D rejects w, and our guess is incorrect. So, $D(w) = 0$, and $\neg w_i = y_i$, which we denote $\neg\mathsf{cg}$

We decompose the "predictor correct" event PC as follows:

$$\Pr[\mathsf{PC}] = \Pr[D(w) \cap \mathsf{cg}] + \Pr[\neg D(w) \cap \neg\mathsf{cg}]$$

We'll transform this expression until we have it in terms of a gap between p_i values, which we have a bound on. Begin by rewriting the joint events, conditioning on correctness of the guess bit:

$$\Pr[\mathsf{PC}] = \Pr[D(w)|\mathsf{cg}] \cdot \Pr[\mathsf{cg}] \ + \ \Pr[\neg D(w)|\neg\mathsf{cg}] \cdot \Pr[\neg\mathsf{cg}]$$

Negate the second event, to get everything in terms of the probability that $D(w) = 1$. This will let us phrase both these events in terms of p_i and p_{i-1}.

$$\Pr[\mathsf{PC}] = \Pr[D(w)|\mathsf{cg}] \cdot \Pr[\mathsf{cg}] \ + \ (1 - \Pr[D(w)|\neg\mathsf{cg}]) \cdot \Pr[\neg\mathsf{cg}]$$

The guess is a bit flipped uniformly at random, so:

$$\Pr[\mathsf{PC}] = (1/2) \cdot \Pr[D(w)|\mathsf{cg}] \ + \ (1/2) \cdot (1 - \Pr[D(w)|\neg\mathsf{cg}]) \qquad (*)$$

Now we relate the string that D is invoked on to hybrids, so we can get the expression (*) in terms of a distinguishing gap. The string w is clearly a sample from $\mathscr{W}_{i-1}$, because we created it by concatenating our input $\vec{y}$ ($i-1$ bits of G-output) with $L-(i-1)$ uniformly random bits. So we can relate p_{i-1} to p_i, by first conditioning on cg.

$$p_{i-1} = \Pr[D(\mathscr{W}_{i-1})] \geq (1/2)\Pr[D(w)|\mathsf{cg}] + (1/2)\Pr[D(w)|\neg\mathsf{cg}]$$

Observe that $\Pr[D(w)|\mathsf{cg}]$ is exactly p_i, because we conditioned on the guess bit being correct — this means that bit i of w is equal to the ith generator output, and so w (conditioned on cg) is actually a sample from $\mathscr{W}_i$. So the above is:

$$p_{i-1} = (1/2)p_i + (1/2)\Pr[D(w)|\neg\mathsf{cg}]$$

Re-arranging, we have that:

$$(1/2)\Pr[D(w)|\neg \mathsf{cg}] = p_{i-1} - (1/2)p_i$$

This is precisely what we wanted: an expression that relates $\Pr[D(w)|\neg \mathsf{cg}]$ to the gap between p_i and p_{i-1}. Now we again use the observation that $\Pr[D(w)|\mathsf{cg}]$ is exactly p_i, substitute the above into (*), and simplify:

$$\begin{aligned}
\Pr[\mathsf{PC}] &\geq (1/2)p_i + (1/2 - (1/2)\cdot \Pr[D(w)|\neg \mathsf{cg}]) \\
&\geq (1/2)p_i + (1/2 - (p_{i-1} - (1/2)p_i)) \\
&\geq (1/2) + (p_i - p_{i-1})
\end{aligned}$$

By the hybrid bound on $p_i - p_{i-1}$ (Lemma 12), this concludes the proof. □

2.4.4 Deterministic but Non-Uniform f-Predictor

We'll begin by constructing a non-uniform predictor for f from $D \in \mathsf{DIS}(G^f, \gamma)$. This is the standard proof presented in textbooks and [NW94]. First, we'll simplify our task using non-uniformity and the probabilistic method. Step (2) of the predictor strategy above selects i, a hybird pair, uniformly at random. Here we only need a non-uniform predictor, so we can skip this step and hardwire in a "good" choice of i, as long as one exists.

Lemma 14 (A Good Hybrid Pair Exists)**.** *If $D \in \mathscr{D}(G, \gamma)$, then there exists $i_\bullet \in [L]$ such that*

$$p_{i_\bullet} - p_{i_\bullet - 1} \geq \gamma/L$$

Proof (by Averaging Argument). Inspect lemma 12. For any random variable Z, if $\mathbb{E}[Z] \geq \rho$, then at least one of the values of Z must be at least ρ. The expectation is over $i \in [L]$, so at least one i much achieve the overall distinguishing gap bound. □

Fix i as guranteed by Lemma 14 above. Then we can construct a deterministic predictor for f. Rewrite the accuracy in Lemma 2 by partitioning the random bits into $\tilde{r}$, which contains the bits r and those bits of z that are *not* named in S_i. Then average over those bits.

$$\Pr_{r,z}[\text{NB}(\vec{y},r) = y_i] = \mathbb{E}_{\tilde{r}}\left[\Pr_{z|_{S_i}}[\text{NB}(\vec{y},r) = y_i|\tilde{r}]\right]$$

Use another averaging argument to fix a choice of $\tilde{r}$ that achieves the guranteed prediction advantage. Finally, observe that $\vec{y}$ is fully determined by the bits of z in $\tilde{r}$ *but for small overlap with* $z|_{S_i}$, by properties of designs. So we can generate any $\vec{y}$ by memorizing a bounded number of f-values and replace $z|_{S_i}$ by input x in the above. This concludes the non-uniform circuit map, which fixes first i and $\tilde{r}$ by averaging, and then uses hardcoded values of f to generate $\vec{y}$ for the next-bit predictor.

Lemma 15. *From a distinguisher for the NW Generator G_f, we can non-uniformly construct a predictor circuit for f.*

2.5 Uniform Construction

Notice that the non-uniformity above was *only* used to obtain f-values and "good" random bits to select a seed and hybrid index. Thus, a randomized algorithm with query access to f (a "teacher") can output, with good probability, a circuit approximately computing f. We outline this algorithm and make some observations about it's computational complexity in the next section.

It consists of a preprocessing stage for fixing a seed and good randomness, and a circuit construction stage that simply operationalizes the reduction to the next-bit predictor given non-uniformly above. The algorithm will, with reasonable probability, print a circuit approximating f, given access to an f-oracle and distinguisher circuit $D \in \mathsf{DIS}(G_f, \gamma)$. To boost the probability of producing a good circuit C, we repeat the reconstruction above $\mathsf{poly}(L)$ times, and estimate, using

random sampling and membership queries to f, the agreement between f and each produced circuit C. We output the best circuit on our list.

PREPROCESSING(i)

1. $r \leftarrow \mathscr{U}_L$ // sample randomness for NB_i
2. $z \leftarrow \mathscr{U}_m$ // sample a "seed" for the PRG
3. For each j in S_i: // erase $z|_{S_i}$
   $z_j \leftarrow \star$
4. for each $j < i$:
   - for each completion $\hat{z}_j$ of $z|_{S_j}$:
     store $f(\hat{z})$ in lookup table T

CIRCUIT-SAMPLER
Pre-process, then build a circuit C from the template:
"On input $x \in \{0,1\}^n$,

1. $z|_{S_i} \leftarrow x$ // obtaining a "completed" seed $z \in \{0,1\}^m$
2. For each $1 \leq j < i$, fix y_j to $f(z|_{S_j})$ by lookup in T.
3. print $\text{NB}(y,r)$"

GENERATE & TEST CIRCUITS
We repeat the two routines above $t = 100L$ times and test to ensure we get a good circuit.

1. $C_i \leftarrow$ **Circuit-Sampler** $10L^2$ times, independently
2. $x_{ij} \leftarrow \mathscr{U}_n$, $10L^2 \times 10L^2$ times
3. $\hat{\alpha}_i \leftarrow (1/t)\sum_j C_i(x_{ij}) == f(x_{ij})$
4. Print C_i with maximal $\hat{\alpha}_i$

Algorithm 3. NW Reconstruction Algorithm $\mathsf{DIS}(G^f, \gamma) \rightarrow \hat{\mathscr{C}}^f_{1/2-1/L}$

Inspecting the procedure above shows it will run in randomized $\mathsf{poly}(L, |D|)$ time and print a circuit $\hat{C}$ computing f with probability $1/2 + O(\gamma/L)$ over the uniform distribution, as promised by Theorem 2.3.1.

Chapter 3

Natural Learning

3.1 Introduction

Circuit analysis problems, problems whose input or output is a Boolean circuit, are a crucial link between designing algorithms and proving lower bounds. For example, Williams [Wil13, Wil14b, Wil14a] shows how to convert non-trivial Circuit-SAT algorithms into circuit lower bounds. In the other direction, there have been many circuit analysis algorithms inspired by circuit lower bound techniques [LMN93, Bra10, San10, ST12, IMP12, IMZ12, BIS12, CKS14, CKK+15, SW15, CK15, CS15, Tal15], but outside the setting of derandomization [NW94, BFNW93, IW97, IKW02, Uma03, KI04], few formal implications giving generic improvements.

Here we make a step towards such generic connections. While we are not able to show that an *arbitrary* way to prove circuit lower bounds yields circuit analysis algorithms, we show that any circuit lower bound proved through the general *natural proofs paradigm* of Razborov and Rudich [RR97] does yield such algorithms. Our main general result is the following.

Theorem 3.1.1 (Main Transfer Theorem, informal version: see Theorem 3.5.1, page 59)**.** *Natural proofs of circuit lower bounds imply learning algorithms for the same circuit class.*

Using known natural lower bounds [Raz87, Smo87, RR97], we get quasipolynomial-time learning algorithms for the hypothesis class $\mathsf{AC}^0[p]$, for any prime p (polynomial-size constant-depth circuits with AND, OR, NOT, and MOD_p gates).

Theorem 3.1.2 (Learning for $\mathsf{AC}^0[p]$: Simplified version, see Corollary 28 page 61)**.** *For every prime $p \geq 2$, there is a randomized algorithm that, given membership queries to an arbitrary n-variate Boolean function $f \in \mathsf{AC}^0[p]$, runs in quasi-polynomial time $n^{\mathsf{poly}\log n}$ and finds a circuit that computes f on all but $1/\mathsf{poly}(n)$ fraction of inputs.*

No learning algorithms for $\mathsf{AC}^0[p]$ were previously known. For AC^0, a learning algorithm was given by Linial, Mansour, and Nisan [LMN93][1], based on Håstad's proof of strong circuit lower bounds for AC^0 [Hås89].

We also apply the general result to immediately obtain the following compression algorithm, first developed (with somewhat stronger parameters) by Srinivasan [Sri15].

Theorem 3.1.3 (Compression for $\mathsf{AC}^0[p]$: Simplified version, see Corollary 29 page 62)**.** *For every prime $p \geq 2$, there is a randomized algorithm that, given the 2^n-bit truth table of an arbitrary n-variate Boolean function $f \in \mathsf{AC}^0[p]$, runs in time $\mathsf{poly}(2^n)$ (polynomial in the input size), and outputs a circuit computing f of the circuit size at most 2^{n-n^μ}, for some $0 < \mu < 1$.*

3.1.1 Compression and learning algorithms from natural lower bounds

Informally, a *natural* lower bound for a circuit class Λ contains an efficient algorithm that distinguishes between the truth tables of "easy" functions (of low Λ-circuit complexity) and those of random Boolean functions. This notion was introduced by Razborov and Rudich [RR97] to capture a common feature of most circuit lower bound proofs: such proofs usually come with efficient algorithms that say something nontrivial about the structure of easy functions in the corresponding circuit class. In [RR97], this observation was used to argue that any circuit class with a natural lower bound is too weak to support cryptography: no strong pseudorandom generator can be computed by a small circuit from the class.

We show that natural circuit lower bounds also imply algorithms for compression and learning of Boolean functions from the same circuit class (provided the circuit class is not too

[1] Their algorithm works in a more general learning model without membership queries, but with access to labeled examples $(x, f(x))$ for uniformly random x.

weak). More precisely, we show how to reduce the task of compressing (learning) Boolean functions in a circuit class Λ to the task of distinguishing between the truth tables of functions of low Λ-circuit complexity and those of random functions. The latter task is exactly what is solved by an efficient algorithm embedded in any natural proof of Λ-circuit lower bounds. Below, we discuss in turn each algorithmic task entailed by Natural Proofs.

Compression. Recall the compression task for Boolean functions: given the truth table of a Boolean function f, print a circuit that computes f. If f is unrestricted, the best guarantee for the circuit size is $2^n/n$ [Lup58, Lup59], and such a circuit can be found in time $\mathsf{poly}(2^n)$, polynomial in the truth table size. We might however be able to do much better for restricted classes of functions. Let Λ be the set of functions computed by some circuit class Λ. Recent work has shown that we can "mine" specific lower bounds against Λ to compress functions $g \in \Lambda$ better than the universal construction [CKK$^+$15]. This work suggested that there should be some generic connection between circuit lower bounds and compression algorithms, but no such connection was known.

We show that any circuit lower bound that is natural in the sense of Razborov and Rudich [RR97] yields a generic compression algorithm for Boolean functions from the same circuit class, provided the circuit class is sufficiently powerful (i.e., containing $\mathsf{AC}^0[p]$ for some prime $p \geq 2$).

A compression algorithm may be viewed as a special case of a natural property: if the compression fails, the function must have high complexity, and compression must fail for most functions. Thus we get an equivalence between these two notions for the case of randomized compression algorithms and BPP-computable natural properties. That is, for an appropriate circuit complexity class $\mathscr{C}$, a BPP-computable natural property against $\mathscr{C}$ implies the existence of a related $\mathscr{C}$-compression algorithm in BPP, and a $\mathscr{C}$-compression algorithm in BPP implies a BPP-computable natural property against $\mathscr{C}$. As our compression algorithms are randomized, we don't get such an equivalence for the case of deterministic natural properties.

Learning. The first stage of our algorithm is a lossy compression of the function in the sense that we get a small circuit that computes the function on *most* inputs. Because this first stage only examines the truth table of the function in relatively few locations, we can view this stage as a *learning algorithm*. This algorithm produces a circuit that approximately computes the given function f with respect to the uniform distribution, and uses membership queries to f. So it fits the framework of PAC learning for the uniform distribution, with membership queries.

Minimum Circuit Size Problem: Search to decision reduction. Our main result also yields a certain "search-to-decision" reduction for the Minimum Circuit Size Problem (MCSP). Recall that in MCSP, one is given the truth table of a Boolean function f, and a parameter s, and needs to decide if the minimum circuit size of f is less than s. Since an efficient algorithm for MCSP would *be* a natural property (with excellent parameters), our main result implies the following: If MCSP is in BPP, then, given oracle access to any n-variate Boolean function f of circuit complexity s, one can find (in randomized polynomial time) a circuit of size $\mathsf{poly}(s)$ that computes f on all but $1/\mathsf{poly}(n)$ fraction of inputs.

3.1.2 Our proof techniques

One of the main tools we use is the Nisan-Wigderson generator construction [NW94], discussed in the previous chapter. Informally, this construction takes as input the truth table of a Boolean function f, and outputs an algorithm for the new function G^f mapping "short" input strings to "long" output strings. The function G^f is intended to be a *pseudo-random generator (PRG)* in the sense that no "small" Boolean circuit can "distinguish" the uniform distribution from the distribution of G^f's outputs (on uniformly random inputs to G^f). A circuit that can distinguish these two distribution is said to break the generator, and is called a *distinguisher*. Nisan and Wigderson [NW94] prove that if the initial function f has "high" circuit complexity, then the function G^f is indeed a PRG. Moreover, their proof is constructive in the sense that there is an efficient *reconstruction* algorithm that, given a distinguisher for G^f and oracle access to f, outputs a "small" Boolean circuit that approximately computes f. See Chapter 2 for the

formal definitions and statements.

Intuitively, we can use this reconstruction algorithm as a *learning algorithm* for a Boolean function f in some circuit class Λ, provided we manage to find an efficient distinguisher for the NW generator G^f. As we shall argue, such a distinguisher for G^f is supplied by any natural proof of Λ-circuit lower bounds (natural property for the circuit class Λ)!

Thus, the main idea of our lossy-compression algorithm is, given the truth table of a Boolean function f from a circuit class Λ,

- imagine using f as the basis for the NW generator G^f,
- argue that the natural property R for the class Λ is a distinguisher for G^f,
- apply the reconstruction algorithm to R to produce a small circuit that approximates f.

For the described approach to work, we need to ensure that (1) there is an efficient reconstruction algorithm that takes a distinguisher for G^f and constructs a small circuit for (approximately computing) f, and (2) the natural property for Λ is a distinguisher for G^f.

For (1), we use the known efficient randomized algorithm that takes a distinguisher for G^f and constructs a small circuit approximately computing f, provided the algorithm is given oracle access to f, described in Chapter 2. The existence of such a uniform algorithm was first observed by Impagliazzo and Wigderson [IW01] (based on [NW94, BFNW93]) in the context of derandomizing BPP under uniform complexity assumptions. Simulating oracle access to f in the framework of [IW01] was quite nontrivial (and required the downward self-reducibility of f). In contrast, we are explicitly given the truth table of f (or allowed membership queries to f), and so oracle access to f is not an issue!

For (2), we must show that each output of the NW generator, when viewed as the truth table of a Boolean function, is computable by a small circuit from the circuit class for which we have a natural lower bound (and so the natural property algorithm can be used as a distinguisher to break the generator). Looking inside the construction of the NW generator, we note that, for a

fixed seed (input) of G^f, each bit of the output of G^f is the value of f on some substring of the seed (chosen via a certain combinatorial structure, the NW design). We argue that the circuit complexity of the truth table output by the NW generator G^f is closely related to the circuit complexity of the original function f.

In particular, we show that if f is in $\mathsf{AC}^0[p]$, and the NW generator has exponential stretch (from $\mathsf{poly}(n)$ bits to 2^{n^γ} bits, for some $\gamma > 0$), then each string output by the NW generator is also a function in $\mathsf{AC}^0[p]$. If, on the other hand, we take the NW generator with certain polynomial stretch, we get that its output strings will be Boolean functions computable by $\mathsf{AC}^0[p]$ circuits of subexponential size. The trade-off between the chosen stretch of the NW generator and the circuit complexity of the string it outputs will be very important for the efficiency of our learning algorithms: the runtime of the learning algorithm will depend polynomially on the stretch of the NW generator. This makes our setting somewhat different than most applications of the NW generator. We will want to make the stretch as small as possible, but must set it above a threshold determined by the quantitative strength of the circuit lower bound that we start from. Thus, the *larger* the circuit size for which we have lower bounds, the *faster* the learning algorithms we get. This is a generic and quantitative way to paraphrase the intuitive observation that the "more" we understand a cirucit class, the "better" we can learn concepts from that class.

Note that if we break the NW generator based on a function f, we only get a circuit that agrees with f on slightly more than half of all inputs. To get a better approximation of f, we employ a standard "hardness amplification" encoding of f, getting a new, amplified function h, and then use h as the basis for the NW generator. The analysis of such hardness amplification is also constructive: it yields an efficient *reconstruction* algorithm that takes a circuit C_0 computing h on more than $1/2$ of the inputs, and constructs a new circuit C that computes the original f on most inputs.

For this amplification to work in our context, we need to ensure that the amplified function h is in the same circuit class as f, and is of related circuit complexity. We show that standard tools such as the Direct Product and XOR constructions have the required properties for $\mathsf{AC}^0[2]$.

For $\mathsf{AC}^0[p]$ where p is prime other than 2, we can't use the XOR construction (as PARITY cannot be computed in $\mathsf{AC}^0[p]$ for any prime $p > 2$ by Smolensky's lower bound [Smo87]). We argue that the MOD_p function can be used for the required amplification within $\mathsf{AC}^0[p]$[2].

Thus, our actual lossy-compression algorithm for a circuit class Λ is as follows:

Given the truth table of a function $f \in \Lambda$,

1. Run the reconstruction algorithm for the NW generator $G^{f^\star}$ with the natural property against Λ as a distinguisher, where $f^\star$ is the amplified version of f. This produces a circuit C_0 computing $f^\star$ on more than $1/2$ of inputs.
2. Run the reconstruction algorithm for hardness amplification to get from C_0 a new circuit C that computes f on most inputs.

To turn this algorithm into an exact compression algorithm, we just patch up the errors by table lookup. Since there are relatively few errors, the size of the patched-up circuit will still be less that the trivial size $2^n/n$.

More interestingly, our lossy compression algorithm described above also yields a *learning* algorithm! The idea is that the reconstruction algorithm for the NW generator G^f runs in time polynomial in the size of the output of the generator, and so only needs at most that many oracle queries to the function f. Rather than being given the full truth table of f, such an algorithm can be simulated with just membership queries to f. Thus we get a learning algorithm with membership queries in the PAC model over the uniform distribution.

Since the runtime of this learning algorithm (and hence also the size of the circuit for f it produces) will be polynomial in the output length of the NW generator that we use to learn f, we would like to minimize the stretch of the NW generator[3]. However, as noted above, *shorter stretch* of the generator means *higher circuit complexity* of the truth table it outputs.

[2]We stress that for our purposes it is important that the *forward direction* of the conditional PRG construction, from a given function f to a generator based on that f, be computable in some low nonuniform circuit class (such as $\mathsf{AC}^0[p]$). In contrast, in the setting of conditional derandomization, it is usually important that the *reverse direction*, from a distinguisher to a small circuit (approximately) computing the original function f, be computable in some low (nonuniform) circuit class (thereby contradicting the assumed hardness of f for that circuit class). One notable exception is hardness amplification within NP [O'D04, HVV06, Tre05].

[3]This is in sharp contrast to the setting of derandomization where one wants to *maximize* the stretch of the generator, as it leads to a more efficient derandomization algorithm.

This in turn means that we need a natural property that works for Boolean functions of higher circuit complexity (i.e., natural properties useful against large circuits). In the extreme case, to learn a polysize Boolean function f in polynomial time, we need to use the NW generator with polynomial stretch, and hence need a natural property useful against circuits of exponential size. In general, there will be a trade-off between the efficiency of our learning algorithm for the circuit class Λ and the usefulness of a natural circuit lower bound for Λ: the larger the size s such that a natural property is useful against Λ-circuits of size s, the more efficient the learning algorithm for Λ.

Razborov and Rudich [RR97] showed that the $\mathsf{AC}^0[p]$ circuit lower bounds due to Razborov [Raz87] and Smolensky [Smo87] can be made into natural properties that are useful against circuits of weakly exponential size 2^{n^γ}, for some $\gamma > 0$ (dependent on the depth of the circuit). Plugging this natural property into our framework, we get our quasi-polynomial-time learning algorithm for $\mathsf{AC}^0[p]$, for any prime p.

We remark that our approach is quite similar to the way Razborov and Rudich [RR97] used natural properties to get new algorithms. They used natural properties to break the cryptographic pseudorandom function generator of [GGM86], which by definition outputs functions of low circuit complexity. Breaking such a generator based on an assumed one-way function F leads to an efficient algorithm for inverting this function F well on average (contradicting the one-wayness of F). We, on the other hand, use the NW generator based on a given function f. The properties of the NW generator construction can be used to show that it outputs (the truth tables of) functions of low circuit complexity, relative to the circuit complexity of f. Thus a natural property for the appropriate circuit complexity class (with an appropriate size parameter) can be used to break this NW generator, yielding an efficient algorithm for producing a small circuit approximating f.

Discussion. One counter-intuitive development in the theory of pseudorandomness has been the prevalence of "win-win" arguments. Typically, in a win-win argument in pseudorandomness, one takes a construction of pseudorandom generator from a hardness assumption

(such as the NW generator mentioned above) and applies it to a function that is *not known* to actually be hard. If the construction is still a PRG, that is a win; if it is not, one learns that the function in question is not hard, and perhaps finds a circuit computing it. Here, we take this paradigm one step further; ours is a "play-to-lose" argument. We apply the pseudorandom generator construction to a function f we *know* not to be hard, in such a way as to guarantee that the resulting generator G^f is *not* pseudorandom. The win in this argument is that the proof of the hardness to pseudorandomness connection gives a way of converting the non-randomness of the generator G^f into a way of computing f, thus translating the knowledge that f is easy to compute into an actual circuit computing f.

3.1.3 Related work

This work was prompted by results that circuit analysis algorithms imply circuit lower bounds. A natural question is: given that these algorithms are *sufficient* for circuit lower bounds, to what degree are they *necessary*? Apart from derandomization, no other equivalences between circuit analysis algorithms and circuit lower bounds are known. Some of the known circuit-analytic algorithmic tasks that would imply circuit lower bounds include: derandomization [IKW02, KI04, AvM12, CIKK15], deterministic (lossy) compression or MCSP [CKK+15, IKW02], deterministic learning [FK09, KKO13], and deterministic (QBF) SAT algorithms [Wil13, SW15].

Bracketing the hardness vs. randomness setting, special cases of using circuit lower bounds to construct circuit analysis algorithms abound. Often, lower bounds are the *only* way that we know to construct these algorithms. Each of the following results uses the proof of a lower bound to construct an algorithm. The character and number of these results gives empirical evidence that there should be generic algorithms for circuit analysis based on generic lower bounds.

- Parity $\notin \mathsf{AC}^0 \rightsquigarrow \mathsf{AC}^0$-Learning [LMN93], AC^0-SAT [IMP12], and AC^0-Compression [CKK+15]

- $\mathrm{MOD}_q \notin \mathsf{AC}^0[p]$, p,q distinct primes, $\rightsquigarrow \mathsf{AC}^0[p]$-Compression [Sri15]

- Andreev's function $\notin \mathrm{deMorgan}[n^{3-\varepsilon}] \rightsquigarrow$ subcubic formula Compression [CKK$^+$15]

All the lower bounds listed above belong to the natural proofs framework. Given these results, the obvious conjecture was that natural proofs imply some kind of generic circuit analysis algorithm. For instance, [CKK$^+$15] suggested that every natural circuit lower bound should imply a compression algorithm. Already, the original work on Natural Proofs observed:

> As long as we use natural proofs we have to cope with a duality: any lower bound proof must implicitly argue a proportionately strong upper bound. With this duality in mind, it is no coincidence that the technical lemmas of [Hås87, Smo87, Raz87] yield much of the machinery for the learning result of [LMN93].
>
> — Razborov and Rudich, 1997

We take a step towards establishing *formal* and *quantitative* duality between lower bound proofs and circuit analysis, by showing that any natural circuit lower bound for a sufficiently powerful circuit class ($\mathsf{AC}^0[p]$ or bigger) does indeed lead to a randomized compression algorithm for the same circuit class. Furthermore, the efficiency of the learning algorithm depends directly on the strength of the lower bound: stronger lower bounds are transformed into faster learning algorithms. Though we are nowhere near an optimal transference theorem from lower bounds to learning algorithms, this is exactly the *type* of dependence that we would hope for.

The remainder of this chapter. We give the necessary background in Section 3.2. Sections 3.3 and 3.4 summarize the useful properties of past constructions of black-box generators and black-box amplifications, which we revisit and modify to implement in $\mathsf{AC}^0[p]$. In Section 3.5, we use those tools to prove our main result that natural properties yield learning algorithms for circuit classes $\mathsf{AC}^0[p]$ and above, using a novel "play-to-lose" interpretation of pseudorandomness. On the other hand, in Section 3.6, we argue that our main result cannot be applied directly to AC^0 because the construction of Section 3.3 is impossible in AC^0. Section 3.7 contains concluding remarks and open questions.

3.2 Definitions and tools

3.2.1 Learning and compression tasks

Let $f \in \Lambda$ be some Boolean function. The learner is allowed membership queries to f. That is, the learner may query an input $x \in \{0,1\}^n$ to the oracle, getting back the value $f(x)$.

Definition 16 (PAC learning over the uniform distribution with membership queries)**.** Let Λ be any class of Boolean functions. An algorithm A PAC-learns Λ if for any n-variate $f \in \Lambda$ and for any $\varepsilon, \delta > 0$, given membership query access to f algorithm A prints with probability at least $1-\delta$ over its internal randomness a circuit $C \in \hat{\mathscr{C}}_{\varepsilon}^{f}$. The runtime of A is measured as a function $T = T(n, 1/\varepsilon, 1/\delta, |f|)$. $\tilde{\mathfrak{C}}$ and $\mathfrak{D}$

Definition 17 (Λ-Compression)**.** Given the truth table of n-variate Boolean function $f \in \Lambda$, print some Boolean circuit $C \in \mathscr{C}^{f}$ computing f such that $|C| < 2^n/n$, the trivial bound.

Definition 18 (ε-Lossy Λ-Compression)**.** Given the truth table of n-variate Boolean function $f \in \Lambda$, print some Boolean circuit $C \in \hat{\mathscr{C}}_{\varepsilon}^{f}$ such that $|C| < 2^n/n$, the trivial bound.

The relevant parameters for compression are runtime and printed circuit size. We say that a compression algorithm is efficient if it runs in time $\mathsf{poly}(2^n)$, which is polynomial in the size of the truth-table supplied to the algorithm. Though we count any output circuit of size less than $2^n/n$ as a successful compression, we will of course want to optimize this. In previous work, the size of the resulting circuits approximately matches the size of circuits for which we have lower bounds.

We remark that we do not obtain "proper" learning or compression: the output of the learning (compression) algorithm is an unrestricted circuit, not necessarily from the class to be learned (compressed).

3.2.2 Natural properties

Let $\mathscr{F}_n$ be the collection of all Boolean functions on n variables. Recall that Λ and Γ denote complexity classes, with Λ generally referring to the weaker or "target" class. A combinatorial property of Boolean functions is a sequence of subsets of $\mathscr{F}_n$ for each n.

Definition 19 (Natural Property [RR97])**.** A combinatorial property $R = \{R_n\}_{n\in\mathbb{N}}$ is Γ-natural against Λ with density $\delta_n \colon \mathbb{N} \to [0,1]$ if it satisfies the following three conditions:

Constructivity: The predicate $f_n \stackrel{?}{\in} R_n$ is computable in Γ

Largeness: $|R_n| \geq \delta_n \cdot |\mathscr{F}_n|$

Usefulness: For any sequence of functions f_n, if $f_n \in \Lambda$ then $f_n \notin R_n$, almost everywhere.

For each n, δ_n is a lower bound on the probability that $g \in \mathscr{F}_n$ has R_n. The original definition in [RR97] sets $\delta_n \geq 2^{-O(n)}$. However, we show (see Lemma 3.2.1 below) that one may usually assume that $\delta_n \geq 1/2$. Note that "in the wild" nearly all natural properties have δ_n close to one and $\Gamma \subseteq \mathsf{NC}^2$.

Lemma 3.2.1 (Largeness for natural properties)**.** *Suppose P is a P-natural property of n-variate Boolean functions that is useful against class Λ of size $s(n)$, and has largeness $\delta_n \geq 2^{-cn}$, for some constant $c \geq 0$. Then there is another P-natural property P' that is useful against the class Λ of size $s'(n) := s(n/(c+1))$, and has largeness $\delta'_n \geq 1/2$.*

The idea is to apply random restrictions to functions that *may* contain (as sub-functions) a truth-table in P. Since restictions are easy to compute, any function that embeds a function from P under restriction is difficult to compute. So usefulness is maintained. Since there are so many random restrictions, density is significantly improved. We work out the details below.

Proof. Define P' as follows:

The truth table of a given $f : \{0,1\}^n \to \{0,1\}$ is in P' iff for at least one string $a \in \{0,1\}^k$, for $k = cn/(c+1)$, the restriction

$$f_a(y_1,\ldots,y_{n-k}) := f(a_1,\ldots,a_k,y_1,\ldots,y_{n-k})$$

is in P (as a function on $n-k = n/(c+1)$ variables).

Observe that testing P' on a given n-variate Boolean function f can be done in time $O(2^k)\cdot \mathsf{poly}(2^{n-k}) \leq \mathsf{poly}(2^n)$; so we have constructivity for P'. Next, if $f : \{0,1\}^n \to \{0,1\}$ has a Λ circuit of size less $s'(n)$, then each restricted subfunction $f_a : \{0,1\}^{n-k} \to \{0,1\}$ has a Λ circuit of size less than $s(n-k) \leq s'(n)$. Finally, a random function $f : \{0,1\}^n \to \{0,1\}$ yields 2^k independent random subfunctions, on $n-k$ variables each, and the probability that at least one of these $(n-k)$-variate functions satisfies P is at least $1-(1-2^{-c(n-k)})^{2^k} = 1-(1-2^{-k})^{2^k}$, which is at least $1/2$, as required. □

3.2.3 NW Generator

Recall the main theorem regarding the NW pseudo-random generator, from Chapter 2:

Theorem 3.2.2 (Uniform NW Reconstruction [NW94, IW01])**.** *There is a randomized algorithm that, given as input a distinguisher circuit D for the NW Generator defined using f and an oracle for f, will print a circuit that approximately computes f with advantage proportional to the stretch of the Generator. Formally:*

$$\mathsf{DIS}(G_f, 1/5) \longmapsto \mathscr{C}^f_{1/2-1/L} \ \textit{in} \ \mathsf{TIME}^f[\mathsf{poly}(L)]$$

3.3 Black-box generators

The main tool we need for our learning algorithms is a transformation, which we call *black-box generator*, taking a given function $f\colon \{0,1\}^n \to \{0,1\}$ to a family $G = \{g_z\}_{z\in I}$ of new Boolean functions $g_z\colon \{0,1\}^{n'} \to \{0,1\}$ satisfying the following properties:

- [NONUNIFORM EFFICIENCY] each function g_z has "small" circuit complexity relative to the circuit complexity of f, and
- [RECONSTRUCTION] any circuit distinguishing a random function g_z (for a uniformly random $z \in I$) from a random n'-variate Boolean function can be used (by an efficient randomized algorithm with oracle access to f) to construct a good approximating circuit for f.

Once we have such a black-box generator, we get our learning algorithm as follows. To learn a function $f\colon \{0,1\}^n \to \{0,1\}$, use the natural property as a distinguisher that rejects (the truth tables of) all functions g_z, $z \in I$, but accepts a constant fraction of truly random functions; apply the efficient reconstruction procedure to learn a circuit approximating f. By nonuniform efficiency, if f is an easy function in some circuit class Λ, then so is each function g_z where $z \in I$.

Next we give a more formal definition of a black-box generator. For a function f, we denote by Λ^f the class of oracle circuits in Λ that have f-oracle gates. Also recall that $\Lambda[s]$ denotes the class of Λ-circuits of size at most s.

Definition 20 (Black-Box (ε, L)-Generator Within Λ)**.** For a given error parameter $\varepsilon\colon \mathbb{N} \to [0,1]$ and a stretch function $L\colon \mathbb{N} \to \mathbb{N}$, a *black-box* (ε, L)*-generator within* Λ is a mapping GEN that associates with a given function $f\colon \{0,1\}^n \to \{0,1\}$ a family $\mathrm{GEN}(f) = \{g_z\}_{z \in \{0,1\}^m}$ of Boolean functions $g_z\colon \{0,1\}^\ell \to \{0,1\}$, where $\ell = \log L(n)$, satisfying the following conditions for every $f : \{0,1\}^n \to \{0,1\}$:

Small Family Size: $m \leq \mathsf{poly}(n, 1/\varepsilon)$,

Nonuniform Λ-Efficiency: for all $z \in \{0,1\}^m$, $g_z \in \Lambda^f[\mathsf{poly}(m)]$, and

Reconstruction: $\mathsf{DIS}(\mathrm{GEN}(f), 1/5) \longmapsto \hat{\mathscr{C}}_\varepsilon^f$ in $\mathsf{BPTIME}^f[\mathsf{poly}(n, 1/\varepsilon, L(n))]$, where we think of $\mathrm{GEN}(f)$ as the distribution over the truth tables of functions $g_z \in \mathrm{GEN}(f)$, for uniformly random $z \in \{0,1\}^m$.

We will prove the following.

Theorem 3.3.1 (Black-Box Generators within $\mathsf{AC}^0[p]$)**.** *Let p be any prime. For every $\varepsilon\colon \mathbb{N} \to [0,1]$ and $L\colon \mathbb{N} \to \mathbb{N}$ such that $L(n) \leq 2^n$, there exists a black-box (ε, L)-generator within $\mathsf{AC}^0[p]$.*

We will use the NW generator as our black-box generator. For it to be within $\mathsf{AC}^0[p]$, we need NW designs to be computable within $\mathsf{AC}^0[p]$. We construct NW designs in $\mathsf{AC}^0[p]$ later — see the proof of Theorem 3.3.5 in Section 3.3.1. This is feasible because we can encode a constant amount of finite-field arithmetic in $\mathsf{AC}^0[p]$, and one of the canonical constructions of designs is simply exhaustive evaluation of polynomials over a finite field.

Theorem 3.3.2 (NW Designs in $\mathsf{AC}^0[p]$)**.** *Let p be any prime. There exists a constant $d_{MX} \geq 1$ such that, for any n and $L < 2^n$, there exists an NW design $S_1, \ldots, S_L \subseteq [m]$ with $m = O(n^2)$, each $|S_i| = n$, and $|S_i \cap S_j| \leq \ell = \log L$ for all $1 \leq i \neq j \leq L$, such that the function $MX_{NW} : \{0,1\}^\ell \times \{0,1\}^m \to \{0,1\}^n$, defined by $MX_{NW}(i,z) = z{\restriction_{S_i}}$, is computable by an $\mathsf{AC}^0[p]$ circuit of size $O(\ell \cdot n^3 \log n)$ and depth d_{MX}.*

Another ingredient we need for the proof of Theorem 3.3.1 is the following notion of black-box amplification; intuitively, a map that is guranteed to transform a mildly hard function into an extremely difficult to compute function. If one can compute the "amplified" function slightly better than random guessing, one can compute the original function extremely accurately. We will exploit this in our learning and compression algorithms. Let Λ be any circuit class.

Definition 21 (Black-Box (ε,δ)-Amplification within Λ)**.**

For given $\varepsilon, \delta > 0$, (ε,δ)-*amplification within* Λ is a mapping that associates with a given function $f\colon \{0,1\}^n \to \{0,1\}$ its *amplified* version, $\textsc{Amp}(f)\colon \{0,1\}^{n'} \to \{0,1\}$, satisfying the following conditions for every $f\colon \{0,1\}^n \to \{0,1\}$:

Short Input: $n' \leq \mathsf{poly}(n, 1/\varepsilon, \log 1/\delta)$,

Nonuniform Λ-Efficiency: $\textsc{Amp}(f) \in \Lambda^f[\mathsf{poly}(n')]$,

Uniform P-Efficiency: $\textsc{Amp}(f) \in \mathsf{P}^f$, and

Reconstruction:

$$\hat{\mathscr{C}}^{\textsc{Amp}(f)}_{1/2-\delta} \longmapsto \hat{\mathscr{C}}^{f}_{\varepsilon} \text{ in } \mathsf{BPTIME}^f[\mathsf{poly}(n, 1/\varepsilon, 1/\delta)]$$

We prove the following in the next section (see Theorems 3.4.3 and 3.4.5).

Lemma 3.3.3. *Let p be any fixed prime. For all $0 < \varepsilon, \delta < 1$, there is black-box (ε, δ)-amplification within $\mathsf{AC}^0[p]$.*

Now we are ready to prove Theorem 3.3.1, by composing Black-Box amplification with the NW designs computable in $\mathsf{AC}^0[p]$ and the NW reconstruction algorithm.

Proof of Theorem 3.3.1. For a given n-variate Boolean function f, consider its amplified version $f^\star = (\varepsilon(n), 1/L(n))$-$\textsc{Amp}(f)$, from the black-box amplification within $\mathsf{AC}^0[p]$ that exists by Lemma 3.3.3. We have that $f^\star$ is a function on $n' = \mathsf{poly}(n, 1/\varepsilon, \log L) = \mathsf{poly}(n, 1/\varepsilon)$ variables (using the assumption that $L(n) \leq 2^n$).

Let $G^{f^\star} \colon \{0,1\}^m \to \{0,1\}^{L(n)}$ be the NW generator based on the function $f^\star$, with the seed size $m = (n')^2$. Define $\textsc{Gen}(f) = \{g_z\}_{z \in \{0,1\}^m}$, where $g_z = G_{f^\star}(z)$. We claim that this $\textsc{Gen}(f)$ is an (ε, L)-black-box generator within $\mathsf{AC}^0[p]$. We verify each necessary property:

Small Family Size: $m = (n')^2 \leq \mathsf{poly}(n, 1/\varepsilon)$.

Nonuniform $\mathsf{AC}^0[p]$-Efficiency: We know that $f^\star = \textsc{Amp}(f) \in (\mathsf{AC}^0[p])^f[\mathsf{poly}(m)]$. For each fixed $z \in \{0,1\}^m$, we have $g_z(i) = (G_{f^\star}(z))_i$, for $i \in \{0,1\}^\ell$, where $\ell = \log L(n)$. By the definition of the NW generator, $g_z(i) = f^\star(z|_{S_i})$. By Theorem 3.3.2, the restriction $z|_{S_i}$, as a function of z and i, is computable in $\mathsf{AC}^0[p]$ of size $\mathsf{poly}(n')$ and some fixed depth d_{MX}. It follows that each g_z is computable in $(\mathsf{AC}^0[p])^f[\mathsf{poly}(m)]$.

Reconstruction: The input to reconstruction is $D \in \mathsf{DIS}(G_{\textsc{Amp}(f)}, 1/5)$. Let $\mathscr{M}_{\mathsf{NW}}$ be the reconstruction machine from the NW construction, and let $\mathscr{M}_{\textsc{Amp}}$ be the reconstruction machine from $(\varepsilon, 1/L)$-amplification. We first run $\mathscr{M}_{NW}^{\textsc{Amp}(f)}(D)$ to get, in time $\mathsf{poly}(L)$, a circuit $C \in \hat{\mathscr{C}}^{\textsc{Amp}(f)}_{1/2-1/L(n)}$; note that we can provide this reconstruction algorithm oracle access to $\textsc{Amp}(f)$,

since $\textsc{Amp}(f) \in \mathsf{P}^f$ by the uniform P-efficiency property of black-box amplification. Next we run $\mathscr{M}^f_{\textsc{Amp}}$ on C to get $C' \in \hat{\mathscr{C}}^f_\varepsilon$, in randomized time $\mathrm{poly}(n, 1/\varepsilon, L(n))$. □

3.3.1 NW designs in $\mathsf{AC}^0[p]$

Here we show that the particular NW designs we need in our compression and learning algorithms can be constructed by small $\mathsf{AC}^0[p]$ circuits, for any fixed prime p. Consider an NW design $S_1, \ldots, S_L \subseteq [m]$, for $m = O(n^2)$, where the sets satisfy the following properties:

- [FIXED SIZE] each set S_i is of size n,
- [WIDE VARIETY] the number of sets is $L = 2^\ell$ for $\ell \leq n$, and
- [BOUNDED OVERLAP] for any two distinct sets S_i and S_j, $i \neq j$, we have $|S_i \cap S_j| \leq \ell$.

We show a particular construction of such a design that has the following property: the index set $[m]$ is partitioned into n disjoint subsets $U_1, \ldots, U_n$ of equal size $(m/n) \in O(n)$. For each $1 \leq i \leq L$, the set S_i contains exactly one element from each subset U_j, over all $1 \leq j \leq n$. For $1 \leq j \leq n$ and $1 \leq k \leq O(n)$, we denote by $(U_j)_k$ the kth element in the subset U_j.

To describe such a design, we use the following Boolean function g: for $1 \leq i \leq L$, $1 \leq j \leq n$, and $1 \leq k \leq O(n)$, we define $g(i, j, k) = 1$ iff $(U_j)_k \in S_i$. We will prove the following.

Theorem 3.3.4 (Local NW Designs in $\mathsf{AC}^0[p]$)**.** *There exists a constant $d_{NW} \geq 1$ such that, for any prime p, there exists a family of functions $g : \{0,1\}^{\ell + 2\log n} \to \{0,1\}$ that are the characteristic functions for some NW design with the parameters as above, so that $g \in \mathsf{AC}^0[p]$ of size $O(n^2 \log n)$ and depth d_{NW}.*

Proof. Recall the standard construction of NW designs from [NW94]. Let F be a field of size $O(n)$. Consider an enumeration of L polynomials of degree at most ℓ over F, with all coefficients in $\{0,1\}$; there are at least $2^\ell = L$ such polynomials. We associate each such polynomial with a

binary string $i = i_1 \dots i_\ell \in \{0,1\}^\ell$, so that i corresponds to the polynomial

$$A_i(x) = \sum_{j=1}^{\ell} i_j \cdot x^{j-1}$$

over the field F. Let $r_1, \dots, r_{|F|}$ be some canonical enumeration of the elements of F. For each binary string $i \in \{0,1\}^\ell$, we define a set $S_i = \{(r_j, A_i(r_j)) \mid 1 \leq j \leq n\}$. Note that $|S_i| = n$, and S_i defines a set of n pairs in the universe $F \times F$ of $O(n^2)$ pairs (hence the universe size for this construction is $O(n^2)$). Finally, any two distinct degree $(\ell - 1)$ polynomials $A_i(x)$ and $A_j(x)$ may agree on at most ℓ points $r \in F$, and so we have $|S_i \cap S_j| \leq \ell$ for the sets S_i and S_j, corresponding to the polynomials $A_i(x)$ and $A_j(x)$.

Arrange the elements of the universe $[m]$ on an $n \times (m/n)$ grid. The n rows of the grid are indexed by the first n field elements $r_1, \dots, r_n$, and the columns by all fields elements $r_1, \dots, r_{|F|}$. For each j, $1 \leq j \leq n$, define U_j to be the elements of $[m]$ that belong to the row j of the grid. We get that every set $S_i = \{(r_j, A_i(r_j)) \mid 1 \leq j \leq n\}$ picks exactly one element from each of the n sets $U_1, \dots, U_n$.

We will argue that this particular design construction is computable in $\mathsf{AC}^0[p]$ of size polynomial in ℓ, for each prime p. Let p be any fixed prime (which we think of as a constant). Let F be an extension field over $\mathbf{GF}(p)$ of the least size so that $|F| \geq n$; such a field is described by some polynomial over $\mathbf{GF}(p)$ of degree $O(\log_p n)$, and is of size at most $pn = O(n)$. As before, let $r_1, \dots, r_{|F|}$ be a canonical enumeration of the field elements in F.

Define the following $n \times \ell$ matrix M: for $1 \leq j \leq n$ and $1 \leq k \leq \ell$, we have $M_{j,k} = (r_j)^k$, where the power $(r_j)^k$ is computed within the field F. Then the values $A_i(r_1), \dots, A_i(r_n)$ may be read off from the column vector obtained by multiplying the matrix M by the column vector $i \in \{0,1\}^\ell$, in the field F. For a particular $1 \leq j \leq n$, we have $A_i(r_j) = \sum_{k=1}^{\ell} M_{j,k} \cdot i_k$. Since each $i_k \in \{0,1\}$, the latter reduces to the task of adding a subset of ℓ field elements. Each field element of F is a polynomial over $\mathbf{GF}(p)$ of degree $k \leq O(\log n)$, and so adding a collection of elements from F reduces to the coordinate-wise summation modulo p of k-element vectors in $(\mathbf{GF}(p))^k$.

The latter task is easy to do in $\mathsf{AC}^0[p]$[4].

For any $1 \le i \le L$, $1 \le j \le n$, and $1 \le k \le |F|$, $g(i,j,k) = 1$ iff $A_i(r_j) = r_k$. To compute $g(i,j,k)$, we need to evaluate the polynomial $A_i(x)$ at r_j, and then check if the result is equal to r_k. To this end, we "hard-code" the matrix M into the circuit (which incurs the cost at most $O(n\ell \log n)$ bits of advice). We compute $A_i(r_j)$ by computing the matrix-vector product $M \cdot i$, and restricting to the jth coordinate of the resulting column vector. This computation involves $O(\log n)$ summations of ℓ field elements of $\mathbf{GF}(p)$ modulo p, over n rows of the matrix M. The resulting field element is described an $O(\log n)$-element vector of elements from the underlying field $\mathbf{GF}(p)$. Using $O(\log n)$ operations over $\mathbf{GF}(p)$, we can check if this vector equals the vector corresponding to r_k.

It is easy to see that this computation can be done in some fixed constant depth d_{NW} by an $\mathsf{AC}^0[p]$ circuit of size $O(\ell \cdot n \log n)$, which can be bounded by $O(n^2 \log n)$, as required. □

As a corollary, we get Theorem 3.3.2, which we restate and prove below.

Theorem 3.3.5 (NW Designs in $\mathsf{AC}^0[p]$, restated)**.** *Let p be any prime. There exists a constant $d_{MX} \ge 1$ such that, for any n and $L < 2^n$, there exists an NW design $S_1, \ldots, S_L \subseteq [m]$ with $m = O(n^2)$, each $|S_i| = n$, and $|S_i \cap S_j| \le \ell = \log L$ for all $1 \le i \ne j \le L$, such that the function $MX_{NW} : \{0,1\}^\ell \times \{0,1\}^m \to \{0,1\}^n$, defined by $MX_{NW}(i,z) = z\restriction_{S_i}$, is computable by an $\mathsf{AC}^0[p]$ circuit of size $O(\ell \cdot n^3 \log n)$ and depth d_{MX}.*

Proof. Let $g(i,j,k)$ be the characteristic function for the NW design from Theorem 3.3.4, where $|i| = \ell$, $|j| = \log n$, and $|k| = \log n + \log c$, for some constant $c \ge 1$. We have $g \in \mathsf{AC}^0[p]$ of size $O(\ell \cdot n \log n)$ and depth d_{NW}. Let $U_1, \ldots, U_n \subseteq [m]$ be the sets of size cn each that partition $[m]$ so that every S_i contains exactly one element from every U_j, $1 \le j \le n$.

[4]We code elements of $\mathbf{GF}(p)$ by p-wire bundles, where wire i is on iff the bundle codes the ith element of $\mathbf{GF}(p)$. An addition, multiplication, or inverse in the field $\mathbf{GF}(p)$ can be implemented in AC^0. To add up a tuple of field elements, we first convert each field element from the representation above to the unary representation (using constant-depth selection circuits). Then we lead these unary encodings into a layer of p gates, $\oplus_p^j$, for $0 \le j \le p-1$, where $\oplus_p^j$ is the gate $\oplus_p$ with $p-j$ extra inputs 1. Thus the gate $\oplus_p^j$ on inputs $x_1, \ldots, x_n \in \mathbf{GF}(p)$ outputs 1 iff $x_1 + \cdots + x_n = j \mod p$. Note that exactly one of the gates $\oplus_p^j$ will output 1, giving us the desired field element in our encoding.

Let $i_1, \ldots, i_\ell$ and $z_1, \ldots, z_m$ denote the input gates of MX_{NW}, and let $y_1, \ldots, y_n$ denote its output gates. Associate each gate y_j with the set U_j of indices in $[m]$, for $1 \leq j \leq n$. For each $1 \leq i \leq L$ and each $1 \leq j \leq n$, define

$$y_j = \vee_{k=1}^{cn} \; g(i,j,k) \wedge (z\restriction_{U_j})_k.$$

Clearly, the defined circuit computes MX_{NW}, because one and only one of the evaluations of g will be true. It has size $O(\ell \cdot n^3 \log n)$ and depth $d_{MX} \leq d_{NW} + 2$, as required. □

Let G^f be the NW generator based on a function f, using the NW design $S_1, \ldots, S_L$ from Theorem 3.3.2. For each fixed seed z, define the function $g_z : \{0,1\}^\ell \to \{0,1\}$, for $\ell = \log L$, as $g_z(i) = (G^f(z))_i = f(z\restriction_{S_i})$, where $1 \leq i \leq L$. By Theorem 3.3.2, we get $g_z \in (\mathsf{AC}^0[p])^f$. See Figure 3.1 for the description of a small circuit for g_z that combines the $\mathsf{AC}^0[p]$ circuit for MX_{NW} with a circuit for f.

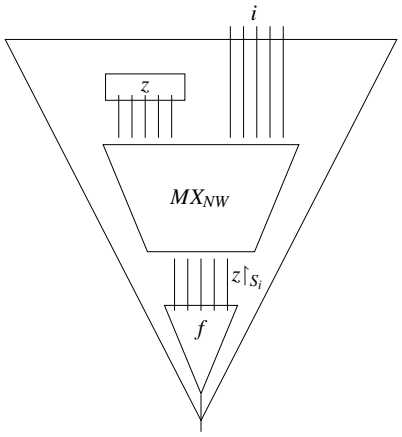

Figure 3.1. A circuit for $g_z(i) = f(z\restriction_{S_i})$.

3.4 Black-box amplification

Here we show that black-box amplification (Definition 21) is possible within $\mathsf{AC}^0[p]$, for any prime $p \geq 2$, as required for the proof that black-box generators within $\mathsf{AC}^0[p]$ exist

(Theorem 3.3.1). Recall the definition:

Let Λ be any circuit class (e.g., $\mathsf{AC}^0[p]$ for some prime $p \geq 2$). For a function f, we denote by Λ^f the class of oracle circuits in Λ that have f-oracle gates. Also recall that $\Lambda[s]$ denotes the class of Λ-circuits of size at most s.

Definition 22 (Black-Box (ε,δ)-Amplification within Λ)**.** For given ε and $\delta > 0$, (ε,δ)-*amplification within* Λ is a mapping that associates with a given function $f : \{0,1\}^n \to \{0,1\}$ its *amplified* version, $\textsc{Amp}(f) : \{0,1\}^m \to \{0,1\}$, satisfying the following conditions:

Short Input: $m \leq \mathsf{poly}(n, 1/\varepsilon, \log 1/\delta)$,

Nonuniform Λ-Efficiency: $\textsc{Amp}(f) \in \Lambda^f[\mathsf{poly}(m)]$,

Uniform P-Efficiency: $\textsc{Amp}(f) \in \mathsf{P}^f$, and

Reconstruction:

$$\hat{\mathscr{C}}^{\textsc{Amp}(f)}_{1/2-\delta} \longmapsto \hat{\mathscr{C}}^{f}_{\varepsilon} \text{ in } \mathsf{BPTIME}[\mathsf{poly}(n, 1/\varepsilon, 1/\delta)]$$

We will show that black-box amplification is possible within $\Lambda = \mathsf{AC}^0[p]$, for any prime $p \geq 2$.

Remark 23. *In our construction, we actually get a better bound on the parameter m: we have* $m \leq O(n \cdot 1/\varepsilon \cdot \log^2(1/\delta))$*, and* $\textsc{Amp}(f) \in \Lambda^f[O(m)]$*. Moreover, we get that there is a fixed constant* $d_{Amp} \geq 1$ *such that (for any prime* $p \geq 2$*) the depth of the* $\mathsf{AC}^0[p]$ *circuit for* $\mathsf{Amp}(f)$ *is at most* d_{Amp} *plus the depth of the* $\mathsf{AC}^0[p]$ *circuit for* f*.*

For $\mathsf{AC}^0[2]$, we shall use standard hardness amplification tools from pseudorandomness: Direct Product and XOR construction. For $\mathsf{AC}^0[p]$, $p \neq 2$, we will need to use something else in place of XOR, as small $\mathsf{AC}^0[p]$ circuits can't compute $\textsc{Parity}$ [Smo87]. We will replace XOR with a MOD_p function, while using an efficient conversion from $\{0,1,\ldots,p-1\}$-valued functions to Boolean functions, which preserves the required amplification parameters.

For a Boolean function $f : \{0,1\}^n \to \{0,1\}$ and a parameter $k \in \mathbb{N}$, the k-wise direct product of f is $f^k : \{0,1\}^{nk} \to \{0,1\}^k$, where $f^k(x_1,\ldots,x_k) = (f(x_1),\ldots,f(x_k))$ for $x_i \in \{0,1\}^n$, $1 \le i \le k$. It is well-known that the Direct Product (DP) construction amplifies hardness of a given function f in the sense that a circuit somewhat nontrivially approximating the function f^k may be used to get a new circuit that approximates the original function f quite well [GNW11], and, moreover, this new circuit for f can be constructed efficiently and uniformly [IW01]. We shall use the following algorithm due to [IJKW10] that has optimal parameters (up to constant factors).

Theorem 3.4.1 (DP Reconstruction [IJKW10])**.** *There is a constant c and a probabilistic algorithm $\mathscr{A}$ with the following property. Let $k \in \mathbb{N}$, and let $0 < \varepsilon, \delta < 1$ be such that $\delta > e^{-\varepsilon k/c}$. For a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$, let C' be any circuit in $\hat{\mathscr{C}}^{f^k}_{1-\delta}$. Given such a circuit C', algorithm $\mathscr{A}$ outputs with probability $\Omega(\delta)$ a circuit $C \in \hat{\mathscr{C}}^{f}_{\varepsilon}$.*

DP Reconstruction Algorithm. The algorithm $\mathscr{A}$ in Theorem 3.4.1 is a uniform randomized NC^0 algorithm (with one C'-oracle gate), and the produced circuit C is an AC^0 circuit of size $\mathsf{poly}(n,k,\log 1/\varepsilon, 1/\delta)$ (with $O((\log 1/\varepsilon)/\delta)$ of C'-oracle gates). We sketch this algorithm below. It consists of a preprocessing stage and a circuit construction stage. For simplicity, we allow the constructed circuit to be randomized; it can easily be made deterministic by choosing all required randomness in the preprocessing stage.

Next, we need to convert a non-Boolean, multi-bit output function $f^k\colon \{0,1\}^{kn} \to \{0,1\}^k$ into a Boolean function h such that a circuit approximately computing h would uniformly and efficiently yield a circuit approximately computing f^k, where the quality of approximation is essentially preserved. To this end, we "collapse" the k-bit output of f^k to a single number modulo a prime p, using the Goldreich-Levin construction [GL89] over $F = \mathbf{GF}(p)$: For $g\colon \{0,1\}^m \to \{0,1\}^k$, define $g^{GL}\colon \{0,1\}^m \times F^k \to F$ to be

$$g^{GL}(x_1,\ldots,x_m,r_1,\ldots,r_k) = \sum_{i=1}^{k} r_i \cdot g(x_1,\ldots,x_m)_i,$$

PREPROCESSING
Randomly pick a set B_0 of k strings in $\{0,1\}^n$. Pick a random subset $A \subset B_0$ of size $k/2$. Evaluate C' on a k-tuple $\vec{b}_0$ that is a random permutation of the strings in B_0, and note the answers $\vec{a}$ given by $C'(\vec{b}_0)$ for the strings in A.

CIRCUIT CONSTRUCTION
Using A and $\vec{a}$ from preprocessing, build a randomized circuit C following the template:
"On input $x \in \{0,1\}^n$, if $x \in A$, then output the corresponding answer in $\vec{a}$. Otherwise, for $m = O((\log 1/\varepsilon)/\delta)$ times,

1. sample a random k-set B such that $A \cup \{x\} \subset B$;
2. evaluate C' on a k-tuple $\vec{b}$ that is a random permutation of the strings in B;
3. if the answers of $C'(\vec{b})$ for A are consistent with $\vec{a}$, then output $C'(\vec{b})_x$ (the answer C' gave for x), and stop.

If no output is produced after m iterations, output a random bit."

Algorithm 4. DP Reconstruction: $\hat{\mathscr{C}}^{f^k}_{1-\delta} \longmapsto \hat{\mathscr{C}}^{f}_{\varepsilon}$

where all arithmetic is over the field F.

We will describe an efficient reconstruction algorithm that takes a circuit computing the function g^{GL} on more than $1/p+\gamma$ fraction of inputs, for some $\gamma > 0$, and produces a circuit that computes g on more than $\Omega(\gamma^3)$ fraction of inputs. The main ingredient of this algorithm is the following result first proved by Goldreich and Levin [GL89] for the case of $p = 2$, and later generalized by Goldreich, Rubinfeld, and Sudan [GRS00] to all primes p.

Theorem 3.4.2 (GL Reconstruction [GL89, GRS00])**.** *There is a probabilistic algorithm $\mathscr{A}$ with the following property. Let $h \in F^k$ be arbitrary, and let $B\colon F^k \to F$ be such that $\mathbf{Pr}_{r\in F^k}[B(r) = \langle h,r\rangle] \geq 1/p+\gamma$ for some $\gamma > 0$, where $\langle x,y\rangle = \sum_{i=1}^{k} x_i \cdot y_i \mod p$. Then, given oracle access to B and the parameter γ, the algorithm $\mathscr{A}$ runs in time $\mathsf{poly}(k, 1/\gamma)$ and outputs a list of size $O(1/\gamma^2)$ such that, with probability at least $1/2$, the tuple h is on the list.*

GL Reconstruction Algorithm. We sketch below the algorithm $\mathscr{A}$ of Theorem 3.4.2.

Proceed in k rounds, maintaining after round i a list $\mathscr{H}_i$ of length-i tuples in F^i; the list after round k is the final output. In round i:

1. Extend each tuple in $\mathscr{H}_{i-1}$ by one element in all $|F|$ possible ways.
2. For each extended tuple $\vec{c} \in F^i$, include $\vec{c}$ in $\mathscr{H}_i$ iff it passes the following test:

   Randomly pick $m = \mathsf{poly}(k/\gamma)$ tuples $\vec{s}_1, \ldots, \vec{s}_m \in F^{k-i}$. For each $\vec{s}_i$ and each $\sigma \in F$, estimate $\mathbf{Pr}_{\vec{r} \in F^i}[B(\vec{r}, \vec{s}) = \langle \vec{c}, \vec{r} \rangle + \sigma]$. If at least one of these estimates is significantly larger than $1/p$, then accept; otherwise, reject.

Algorithm 5. GL Reconstruction (sketch)

3.4.1 Case of $\mathsf{AC}^0[2]$

Composing Direct-Product and Goldreich-Levin reconstrction (Theorems 3.4.1 and 3.4.2) we have the following.

Theorem 3.4.3 (Black-Box Amplification within $\mathsf{AC}^0[2]$)**.** *For any $0 < \varepsilon, \gamma < 1$, there is black-box (ε, γ)-amplification within $\mathsf{AC}^0[2]$.*

Proof. Given $f\colon \{0,1\}^n \to \{0,1\}$ in $\mathsf{AC}^0[2]$ of size s, and given $0 < \varepsilon, \delta < 1$, define $\textsc{Amp}(f)$ as follows:

1. Set $k = \lceil (3c) \cdot 1/\varepsilon \cdot \ln 1/\gamma \rceil + 1$, where c is the constant in Theorem 3.4.1.
2. Define g to be the direct product $f^k\colon \{0,1\}^{nk} \to \{0,1\}^k$.
3. Define $\textsc{Amp}(f)$ to be $g^{GL}\colon \{0,1\}^{nk+k} \to \{0,1\}$ over $F = \mathrm{GF}(2)$.

Notice that GL reconstruction is an oracle-algorithm, but we need to print a concrete circuit good for many inputs. So we pause to show that we can do exactly this by "baking" GL reconstruction into a circuit in the natural way:

Claim 24 (GL Reconstruction can build circuits instead.)**.** *For any $\gamma > 0$, we have*

$$\hat{\mathscr{C}}^{g^{GL}}_{1/2-\gamma} \longmapsto \hat{\mathscr{C}}^{g}_{1-\Omega(\gamma^3)} \text{ in } \mathsf{BPTIME}^f[\mathsf{poly}(n, k, 1/\gamma)]$$

Proof. Suppose we are given a circuit $C' \in \hat{\mathscr{C}}^{g^{GL}}_{1/2-\gamma}$. Let $\mathscr{A}_{GL}$ be the Goldreich-Levin algorithm of Theorem 3.4.2. Consider the following algorithm $\mathscr{A}_1$ that attempts to compute g:

> For a given input $x \in \{0,1\}^{nk}$, define a circuit $B_x(r) := C'(x,r)$, for $r \in \{0,1\}^k$. Run $\mathscr{A}_{GL}$ on B_x, with parameter $\gamma/2$, getting a list L of k-bit strings. Output a uniformly random k-bit string from the list L.

CORRECTNESS ANALYSIS OF $\mathscr{A}_1$: By averaging, for each of at least $\gamma/2$ fraction of strings $x \in \{0,1\}^{nk}$, the circuit $B_x(r) := C'(x,r)$ agrees with $g^{GL}(x,r) = \langle g(x), r\rangle$ on at least $1/2+\gamma/2$ fraction of strings $r \in \{0,1\}^k$. For each such x, the circuit B_x satisfies the condition of Theorem 3.4.2, and so the GL algorithm will find, with probability at least $1/2$, a list L of $O(1/\gamma^2)$ strings in $\{0,1\}^k$ that contains the string $g(x)$. Conditioned on the list containing the string $g(x)$, if we output a random string on that list, we get $g(x)$ with probability at least $1/|L| \geq \Omega(\gamma^2)$. Overall, the fraction of inputs x where $\mathscr{A}_1$ correctly computes $g(x)$ is at least $\frac{\gamma}{2} \cdot \frac{1}{2} \cdot \Omega(\gamma^2) \geq \Omega(\gamma^3)$. The runtime of $\mathscr{A}_1$ is $\mathsf{poly}(|C'|, k, n, 1/\gamma)$. □

By the Direct Product reconstruction of Theorem 3.4.1, we have:

$$\hat{\mathscr{C}}^{f^k}_{1-\mu} \longmapsto \hat{\mathscr{C}}^{f}_{\varepsilon} \text{ in } \mathsf{BPTIME}^f[\mathsf{poly}(n, k, \log 1/\varepsilon, 1/\mu)]$$

as long as $\mu > e^{-\varepsilon k/c}$, for some fixed constant $c > 0$. Combining this with the "circuit-baked" GL reconstruction from Claim 24 above yields

$$\hat{\mathscr{C}}^{\text{AMP}(f)}_{1/2-\gamma} \longmapsto \hat{\mathscr{C}}^{f}_{\varepsilon} \text{ in } \mathsf{BPTIME}^f[\mathsf{poly}(n, 1/\varepsilon, 1/\gamma)]$$

as long as $\gamma^3 > e^{-\varepsilon k/c}$, which is equivalent to $\gamma > e^{-\varepsilon k/c'}$, for $c' = 3c$. Our choice of k satisfies this condition.

Finally, we verify that $\text{AMP}(f)$ also satisfies the other conditions of black-box amplification:

- $(f^k)^{GL}$ is defined on inputs of size $kn + k \leq O(n \cdot 1/\varepsilon \cdot \log 1/\gamma)$.

- If $f \in \mathsf{AC}^0[2]$ of size s, then f^k is in $\mathsf{AC}^0[2]$ of size $O(s \cdot k) = O(s \cdot 1/\varepsilon \cdot \log 1/\gamma)$, and $(f^k)^{GL}$ is of size at most the additive term $O(k)$ larger.

- $(f^k)^{GL}$ is in P^f.

Thus, $\textsc{Amp}(f)$ defined above is black-box (ε, γ)-amplification of f, as required. □

3.4.2 Case of $\mathsf{AC}^0[p]$ for primes $p > 2$

For $\mathsf{AC}^0[p]$ circuits, with $p > 2$, we can't use the XOR construction above, as PARITY is not computable by small $\mathsf{AC}^0[p]$ circuits [Smo87]. A natural idea to amplify a given function f is to apply the Goldreich-Levin construction g^{GL} over the field $F = \mathrm{GF}(p)$ to the direct-product function $g = f^k$, for an appropriate value of k. Theorem 3.4.2 guarantees that if we have a circuit that computes g^{GL} on more than $1/p + \gamma$ fraction of inputs, then we can efficiently construct a circuit that computes g on $\Omega(\gamma^3)$ fraction of inputs; the proof is identical to that of Claim 24 inside the proof of Theorem 3.4.3 for the case of $\mathsf{AC}^0[2]$ above.

The only problem is that the function g^{GL} defined here is *not* Boolean-valued, but we need a Boolean function to plug into the NW generator in order to complete our construction of a black-box generator within $\mathsf{AC}^0[p]$. We need to convert g^{GL} into a Boolean function h in such a way that if h can be computed by some circuit on at least $1/2 + \mu$ fraction of inputs, then g^{GL} can be computed by a related circuit on at least $1/p + \mu'$ fraction of inputs, where μ and μ' are close to each other.

We use von Neumann's idea for converting a coin of unknown bias into a perfectly unbiased coin [vN51]. Given a coin that is "heads" with some (unknown) probability $0 < p < 1$, flip the coin twice in a row, independently, and output 0 if the trials were ("heads", "tails"), or 1 if the trials were ("tails", "heads"). In case both trials came up the same (i.e., both "heads", or both "tails"), flip the coins again.

Observe that, conditioned on producing an answer $b \in \{0,1\}$, the value b is uniform over $\{0,1\}$ (as both conditional probabilities are equal to $p(1-p)/(1-p^2-(1-p)^2)$. The

probability of not producing an answer in one attempt is $p^2 + (1-p)^2$, the collision probability of the distribution $(p, 1-p)$. If p is far away from 0 and 1, the probability that we need to repeat the flipping for more than t trials diminishes exponentially fast in t.

In our case, we can think of the value of g^{GL} on a uniformly random input as a distribution over F. Assuming that this distribution is close to uniform over F, we will define a new Boolean function h based on g^{GL} so that the output of h on a uniformly random input is close to uniform over $\{0,1\}$. Our analysis of h will be constructive in the following sense. If we are given a circuit that distinguishes the distribution of the outputs of h from uniform, then we can efficiently construct a circuit that distinguishes the distribution of the outputs of g^{GL} from uniform over F. Finally, using the standard tools from pseudorandomness (converting distinguishers into predictors), we will efficiently construct from this distinguisher circuit a new circuit that computes g^{GL} on noticeably more than $1/p$ fraction of inputs.

The construction of this function h follows the von Neumann trick above. Formally we have the following.

Definition 25 (von Neumann trick function)**.** For an integer parameter $t > 0$, define the function $E^{vN} \colon (F^2)^t \to \{0,1\}$ as follows: For pairs $(a_1,b_1),\ldots,(a_t,b_t) \in F \times F$, set

$$E^{vN}((a_1,b_1),\ldots,(a_t,b_t)) = \begin{cases} 1 & \text{if, for each } 1 \le i \le t,\ a_i = b_i \\ 1 & \text{if } (a_i,b_i) \text{ is the first unequal pair and } a_i > b_i \\ 0 & \text{if } (a_i,b_i) \text{ is the first unequal pair and } a_i < b_i \end{cases}$$

It is not hard to see that E^{vN} is computable in AC^0. Moreover, for independent uniformly distributed inputs, the output of E^{vN} is a random coin flip, with bias at most $(1/p)^t$.

Claim 26 (von Neumann trick is small-bias)**.** *Let $\mathscr{F}$ be the uniform distribution over the field $F = \mathbf{GF}(p)$, and let $\mathscr{G} = (\mathscr{F}^2)^t$ be the uniform distribution over sequences of t pairs of elements*

from F. Then:

$$\left|\mathbf{Pr}_{r\in\mathscr{G}}[E^{vN}(r)=1]-\mathbf{Pr}_{r\in\mathscr{G}}[E^{vN}(r)=0]\right|\leq p^{-t}$$

Proof. Conditioned on having some unequal pair in the sample from $\mathscr{G}$, the bias of the random variable $E^{vN}(\mathscr{G})$ is 0. Conditioned on having no such unequal pair, the bias is at most 1. Note that the collision probability of the uniform distribution over $\mathbf{GF}(p)$ is $\sum_{i=1}^{p} p^{-2} = p^{-1}$. So the probability of having collisions in all t independent samples from $\mathscr{F}^2$ is p^{-t}. Thus, the overall bias is at most p^{-t}. □

Next, given $g^{GL}\colon D\to F$, for the domain $D=\{0,1\}^m\times F^k$, define $h^{vN}\colon (D^2)^t\to\{0,1\}$ as follows:

$$h^{vN}((a_1,b_1),\ldots,(a_t,b_t)) = E^{vN}((g^{GL}(a_1),g^{GL}(b_1)),\ldots,(g^{GL}(a_t),g^{GL}(b_t))).$$

Theorem 3.4.4 (vN_p Reconstruction $\implies$ GL_p Reconstruction)**.** *For any $0<\mu<1$ and $1>\gamma>\Omega(\mu/(\log 1/\mu))$, we have*

$$\hat{\mathscr{C}}^{h^{vN}}_{1/2-\mu}\longmapsto\hat{\mathscr{C}}^{g^{GL}}_{1-1/p-\gamma}\ \textit{in}\ \mathsf{BPTIME}^f[\mathsf{poly}(k,m,\mathsf{poly}(1/\mu))]$$

Proof. Recall some basic definition from pseudorandomness theory. We say that distributions X and Y are computationally (η,s)-indistinguishable, denoted by $X\overset{\eta,s}{\approx}Y$ if, for any circuit T of size s, the probability that T accepts a sample from X is the same as the probability T accepts a sample from Y, to within $\pm\eta$.

We want to show that if h^{vN} is predictable with probability better than $1/2$, then g^{GL} is predictable with probability better than $1/p$. We will argue the contrapositive: suppose g^{GL} is unpredictable, and show that h^{vN} is unpredictable. This will take a sequence of steps.

Let $\mathscr{D}$ denote the uniform distribution over D, $\mathscr{F}$ the uniform distribution over F, and $\mathscr{U}$ the uniform distribution over $\{0,1\}$. Assume g^{GL} is unpredictable by circuits of size s

with probability better than $1/p+\gamma$, for some $\gamma > 0$. This implies the following sequence of statements:

1. $(\mathscr{D}, g^{GL}(\mathscr{D})) \overset{2\gamma,\Omega(s)}{\approx} (\mathscr{D}, \mathscr{F})$ (unpredictable $\Rightarrow$ indistinguishable)
2. $(\mathscr{D}^{2t}, g^{GL}(\mathscr{D})^{2t}) \overset{4t\gamma,\Omega(s/t)}{\approx} (\mathscr{D}^{2t}, F^{2t})$ (hybrid argument)
3. $(\mathscr{D}^{2t}, E^{vN}(g^{GL}(\mathscr{D})^{2t})) \overset{4t\gamma,\Omega((s/t)-\mathsf{poly}(t))}{\approx} (\mathscr{D}^{2t}, E^{vN}(F^{2t}))$ (applying h^{vN})
4. $(\mathscr{D}^{2t}, h^{vN}(\mathscr{D}^{2t})) \overset{4t\gamma+p^{-t},\Omega((s/t)-\mathsf{poly}(t))}{\approx} (\mathscr{D}^{2t}, \mathscr{U})$ (by Claim 26)

Finally, the last statement implies (via the "indistinguishable to unpredictable" direction) that h^{vN} cannot be computed on more than $1/2+\mu$ fraction of inputs by any circuit of size $\Omega((s/t)-\mathsf{poly}(t))$, where $\mu = \Omega(t\gamma+p^{-t})$. For $t = O(\log 1/\mu)$, we get $\gamma \geq \Omega(\mu/(\log 1/\mu))$.

In the standard way, the sequence of implications above yields an efficient randomized algorithm, with the runtime $\mathsf{poly}(k,m,\log 1/\mu)$, for going in the reverse direction: from a predictor circuit for h^{vN} to a predictor circuit for g^{GL}. To be able to carry out the hybrid argument with uniform algorithms, we need efficient sampleability of the distribution $(\mathscr{D}, g^{GL}(\mathscr{D}))$. Such sampling is possible when we have membership queries to f (as $g^{GL} \in \mathsf{P}^f$); in fact, here it would suffice to have access to uniformly random labeled examples $(x, f(x))$. Another issue is that we need to sample uniformly from $\mathbb{Z}_p$, while we only have access to uniformly random bits. However, it is easy to devise an efficient sampling algorithm for $\mathbb{Z}_p$, with the distribution statistically almost indistinguishable from uniform over $\mathbb{Z}_p$.[5] □

We now have all the ingredients to prove the following.

Theorem 3.4.5 (Black-Box Amplification within $\mathsf{AC}^0[p]$)**.** *For any $0 < \varepsilon, \gamma < 1$, there is black-*

[5]We divide an interval $[0, 2^{k-1}]$ into p almost equal pieces (all but the last piece are equal to $\lfloor 2^k/p \rfloor$), and check in AC^0 which piece we fall into. The statistical difference between the uniform distribution over $\mathbb{Z}_p$ and this distribution is at most $p/2^k$. So we can make it negligible by choosing k to be a large enough polynomial in the relevant parameters.

box (ε,γ)*-amplification within* $\mathsf{AC}^0[p]$.

Proof (sketch). The proof is similar to that of Theorem 3.4.3. To amplify a given function f, we first apply the Direct Product construction to get $g = f^k$ (for an appropriate parameter k), then use the Goldreich-Levin construction to get g^{GL}, and finally apply the von Neumann construction h^{vN}. The only difference is the use of the von Neumann construction of Theorem 3.4.4. But the only consequence of this extra step for the parameters of the amplification procedure is the slightly worse dependence on $1/\gamma$: from $1/\gamma$ to $(1/\gamma)\cdot\log 1/\gamma \leq 1/\gamma^2$. □

For the remainder of this section, we give deferred proofs of standard results in pseudo-randomness or complexity lower bounds on standard tricks. These lemmas can be skipped on a first reading.

Yao's "Distinguisher to Predictor" reduction

The following result is a simple generalization of Yao's "distinguisher to predictor" reduction for the case of non-binary alphabets. We give the proof as we could not find a reference in the literature for this version of the result.

Lemma 3.4.6 (Yao)**.** *Let* $f\colon \{0,1\}^n \to \mathbb{Z}_p$*. Suppose there is a function* $T\colon \{0,1\}^n \times \mathbb{Z}_p \to \{0,1\}$ *such that*

$$\mathbf{Pr}_{x\in\{0,1\}^n}[T(x,f(x))=1] - \mathbf{Pr}_{x\in\{0,1\}^n,g\in\mathbb{Z}_p}[T(x,g)=1] \geq \varepsilon, \tag{3.1}$$

then the following algorithm P computes f with probability at least $1/p+\varepsilon/(p-1)$ *with respect to the uniform distribution over* $\{0,1\}^n$*:*

> *On input* $x\in\{0,1\}^n$*, pick a uniformly random* $g\in\mathbb{Z}_p$*. Compute* $b = T(x,g)$*. If* $b=1$*, then output g; otherwise, output a uniformly random* $g'\in\mathbb{Z}_p\setminus\{g\}$*.*

Proof. Using Bayes's formula, the probability that the algorithm P above is correct on a uniformly random $x\in\{0,1\}^n$, $\mathbf{Pr}_x[P(x)=f(x)]$, can be written as the sum of the following two expressions:

$$\mathbf{Pr}_{x,g}[T(x,g)=1 \mid g=f(x)]\cdot\mathbf{Pr}_{x,g}[g=f(x)], \tag{3.2}$$

and

$$\mathbf{Pr}_{x,g,g'}[T(x,g)=0 \;\&\; g'=f(x) \mid g \neq f(x)] \cdot \mathbf{Pr}_{x,g}[g \neq f(x)] \tag{3.3}$$

where x is a uniformly random sample from $\{0,1\}^n$, g is a uniformly random sample from $\mathbb{Z}_p$, and g' is uniform over the set $\mathbb{Z}_p \setminus \{g\}$. Since g is independent of x, we have $\mathbf{Pr}_{x,g}[g=f(x)] = 1/p$. Thus we can replace the last factor in Eq. (3.2) by $1/p$, and the last factor in Eq. (3.3) by $(p-1)/p$.

Next, applying Bayes's formula to the first factor of Eq. (3.3), we can re-write this factor as

$$\mathbf{Pr}_{x,g}[T(x,g)=0 \mid g \neq f(x)] \cdot \mathbf{Pr}_{x,g'}[g'=f(x) \mid g \neq f(x),\ T(x,g)=0],$$

which equals

$$\mathbf{Pr}_{x,g}[T(x,g)=0 \mid g \neq f(x)] \cdot \frac{1}{p-1} \tag{3.4}$$

(since $f(x) \neq g$ and $g' \in \mathbb{Z}_p \setminus \{g\}$ is independent of x and g).

Putting Eqs. (3.2)–(3.4) together, we get

$$\begin{aligned}\mathbf{Pr}_x[P(x)=g(x)] &= \frac{1}{p} \cdot (\mathbf{Pr}_x[T(x,f(x))=1] + \mathbf{Pr}_{x,g}[T(x,g)=0 \mid g \neq f(x)]) \\ &= \frac{1}{p} \cdot (\mathbf{Pr}_x[T(x,f(x))=1] + (1 - \mathbf{Pr}_{x,g}[T(x,g)=1 \mid g \neq f(x)])) .\end{aligned}$$

So we have

$$\mathbf{Pr}_x[P(x)=g(x)] = \frac{1}{p} + \frac{1}{p} \cdot (\mathbf{Pr}_x[T(x,f(x))=1] - \mathbf{Pr}_{x,g}[T(x,g)=1 \mid g \neq f(x)]) . \tag{3.5}$$

On the other hand, we have

$$\mathbf{Pr}_{x,g}[T(x,g)=1] = \frac{1}{p} \cdot \mathbf{Pr}[T(x,f(x))=1] + \left(1 - \frac{1}{p}\right) \cdot \mathbf{Pr}[T(x,g)=1 \mid g \neq f(x)].$$

Therefore, we get

$$\begin{aligned}&\mathbf{Pr}[T(x,f(x))=1]-\mathbf{Pr}[T(x,g)=1]\\&\qquad=\frac{p-1}{p}\cdot\left(\mathbf{Pr}[T(x,f(x))=1]-\mathbf{Pr}[T(x,g)=1\mid g\neq f(x)]\right),\end{aligned}\tag{3.6}$$

and so, using Eq. (3.1), we have

$$\mathbf{Pr}[T(x,f(x))=1]-\mathbf{Pr}[T(x,g)=1\mid g\neq f(x)]\geq\frac{p}{p-1}\cdot\varepsilon.\tag{3.7}$$

Plugging in Eq. (3.7) into Eq. (3.5), we conclude

$$\mathbf{Pr}[P(x)=f(x)]\geq\frac{1}{p}+\frac{\varepsilon}{p-1},$$

as required. □

The von Neumann function in AC^0

It is straightforward to implement the von Neumann trick in AC^0 by using the Descriptive Complexity framework to write uniform AC^0 circuits as formulas of first-order logic (FO) over finite models. Specifically, DLOGTIME-uniform AC^0 is captured by FO-formulas over finite models equipped with the following relations: $\{=,<,+,\times,BIT\}$. For details on how first-order logic corresponds to circuit classes, see [BIS90].

To code a length-n string of coinflips $s\in\{H,T\}^n$ as a finite model, we think of the universe set as representing positions or indices into s. Speficially, start with a size-n model equipped with the "default" relations $\{=,<,+,\times,BIT\}$, where the universe elements are interpreted as the n-initial prefix of $\mathbb{N}$ for the purpose of these relations. Then add the following unary relations to represent the character at each position of s:

$$H(i) = \begin{cases} \text{true} & \text{if } s[i] = H \\ \text{false} & \text{otherwise} \end{cases}$$

$$T(i) = \begin{cases} \text{true} & \text{if } s[i] = T \\ \text{false} & \text{otherwise} \end{cases}$$

See definition 25 for a complete description of E^{vN}, the von Neumann trick function. Using the above coding of strings into finite models, we prove the following:

Lemma 3.4.7. $E^{vN} \in \mathsf{AC}^0$

Proof. We write the von Neumann trick as a FO forumla by considering, for the t trials, $2t$-size finite models equipped with the heads and tails relations. Our objective is to detect if the first mismatched pair of indicies is HT or TH, returning true if this pair is TH. We consider a trial to have failed if both flips match. So our formula should say "the first trial that didn't fail is TH ". For now, assume that the following predicates TRIAL and FAIL are FO-expressible:

$$TRIAL(i,j) = \begin{cases} \text{true} & \text{if } (i,j) \text{ are a trial pair with } i < j \\ \text{false} & \text{otherwise} \end{cases}$$

$$FAIL(i,j) = \begin{cases} \text{true} & \text{if } (i,j) \text{ are a failed TRIAL pair} \\ \text{false} & \text{otherwise} \end{cases}$$

We can use FO to detect the first useful trial by asserting that every previous trial failed:

$$\exists i,j(TRIAL(i,j) \wedge T(i) \wedge H(j) \wedge \forall k,\ell\ (k < i \wedge \ell < j \implies FAIL(k,\ell)))$$

This formula is true if and only if the first useful coinflip is TH. Otherwise, if the first

useful trial is HT or there are no useful trials, it is false. This is exactly the behavior we want to implement the the von Neumann trick. All that remains is to give FO formulas for TRIAL and FAIL. These are straightforward, because they involve only simple arithmetic on indices of the string and we have built-in numeric predicates for this. □

3.5 Natural properties imply randomized learning

In this section, we prove the general implication from natural properties to learning algorithms. First we prove the generic reduction from learning (and compression) to natural properties. Then, as our main application, we use the known natural properties for $\mathsf{AC}^0[p]$, to get learning and compression algorithms for $\mathsf{AC}^0[p]$.

3.5.1 A generic reduction from learning to natural properties

Theorem 3.5.1 (Learning from a natural property)**.** *Let Λ be any circuit class containing $\mathsf{AC}^0[p]$ for some prime p. Let R be a P-natural property, with largeness at least $1/5$, that is useful against $\Lambda[u]$, for some size function $u\colon \mathbb{N}\to\mathbb{N}$. Then there is a randomized algorithm that, given oracle access to any function $f\colon \{0,1\}^n \to \{0,1\}$ from $\Lambda[s_f]$, produces a circuit $C \in \hat{\mathscr{C}}_\varepsilon^f$ in time $\mathsf{poly}(n, 1/\varepsilon, 2^{u^{-1}(\mathsf{poly}(n,1/\varepsilon,s_f))})$.*

Proof. Let $\textsc{Gen}(f) = \{g_z\}$ be an (ε, L)-black-box generator based on f, for $L(n)$ such that $\log L(n) > u^{-1}(\mathsf{poly}(n, 1/\varepsilon(n), s_f))$. Using the nonuniform Λ-efficiency of black-box generators, we have that $g_z \in \Lambda^f[\mathsf{poly}(n,1/\varepsilon)]$, for every z. Hence, we get, by replacing the f-oracle with the Λ-circuit for f, that $g_z \in \Lambda[s_g]$, for some $s_g \leq \mathsf{poly}(n, 1/\varepsilon, s_f)$. We want $s_g < u(\log L(n))$. This is equivalent to $u^{-1}(s_g) < \log L(n)$.

Let D be the circuit obtained from the natural property R restricted to truth tables of size $L(n)$. By usefulness, we have $\Pr_z[\neg D(g_z) = 1] = 1$, and by largeness, $\Pr_y[\neg D(y) = 1] \leq 1 - 1/5$. So $\neg D$ is a $1/5$-distinguisher for $\textsc{Gen}(f)$. By the reconstruction property of black-box generators,

we have a randomized algorithm that constructs a circuit $C \in \hat{\mathscr{C}}_{\varepsilon}^{f}$ in time $\mathsf{poly}(n, 1/\varepsilon(n), L(n)) = \mathsf{poly}(n, 1/\varepsilon, 2^{u^{-1}(\mathsf{poly}(n,1/\varepsilon,s_f))})$, as required. □

As discussed in section 3.1.3, stronger circuit lower bounds are transformed into faster learning algorithms. Unsurprisingly, these dependences follow the same pattern as complexity-theoretic hardness-randomness tradeoffs. For different usefulness bounds u, we get different runtimes for our learning algorithm:

- polynomial $\mathsf{poly}(ns_f/\varepsilon)$, for $u(n) = 2^{\Omega(n)}$,
- quasi-polynomial $\mathsf{quasi\text{-}poly}(ns_f/\varepsilon)$, for $u(n) = 2^{n^{\alpha}}$ where $\alpha < 1$, and
- subexponential $\mathsf{poly}(n, 1/\varepsilon, 2^{(ns_f/\varepsilon)^{o(1)}})$, for $u(n) = n^{\omega(1)}$.

Corollary 27. *Under the same assumptions as in Theorem 3.5.1, we also get randomized compression for* $\Lambda[\mathsf{poly}]$ *to the circuit size at most* $O(\varepsilon(n) \cdot 2^n \cdot n)$*, for any* $0 < \varepsilon(n) < 1$ *such that* $\log(\varepsilon(n) \cdot 2^n \cdot n) \geq u^{-1}(\mathsf{poly}(n, 1/\varepsilon))$.

Proof. We use Theorem 3.5.1 to learn a small circuit that computes f on all but at most $\varepsilon \cdot 2^n$ inputs, and then patch up this circuit by hardwiring all the error inputs, using extra circuitry of size at most $O(\varepsilon \cdot 2^n \cdot n)$. This size will dominate the overall size of the patched-up circuit for the ε satisfying the stated condition. □

3.5.2 Application: Learning and compression algorithms for $\mathsf{AC}^0[p]$

We have natural properties useful against the class of AC^0 circuits with mod p gates, for any fixed prime p, as given in [RR97]. The lower bound of Razborov [Raz87] (showing that MAJORITY is not in $\mathsf{AC}^0[2]$) embeds a natural property against $\mathsf{AC}^0[2]$, and the lower bound of Smolensky [Smo87] (showing that PARITY is not in $\mathsf{AC}^0[p]$, for any prime $p \neq 2$) embeds a natural property against $\mathsf{AC}^0[p]$ for any prime $p > 2$. In both cases, the natural property is NC^2-computable, and is useful for circuit size up to $2^{\Omega(n^{1/(2d)})}$, where d is the circuit depth, and n is the input size.

Theorem 3.5.2 ([RR97])**.** *For every prime p, there is an NC^2-natural property of n-variate Boolean functions, with largeness at least $1/2$, that is useful against $\mathsf{AC}^0[p]$ circuits of depth d of size up to $2^{\Omega(n^{1/(2d)})}$.*

Below we sketch the corresponding natural properties; proofs are deferred to Section 3.5.4, after we have completed our main application.

Natural Property useful against $\mathsf{AC}^0[2]$: For $0 \leq a, b \leq n$, define a linear transformation $A_{a,b}$ that maps a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ to a matrix $M = A_{a,b}(f)$ of dimension $\binom{n}{a} \times \binom{n}{b}$, whose rows are indexed by size a subsets of $[n]$, and rows by size b subsets of $[n]$. For every $K \subseteq [n]$, define the set $Z(K) = \{(x_1, \ldots, x_n) \in \{0,1\}^n \mid \forall i \in K,\ x_i = 0\}$. For a size a subset $I \subseteq [n]$ and size b subset $J \subseteq [n]$, define $M_{I,J} = \oplus_{x \in Z(I \cup J)} f(x)$.

The natural property of Theorem 3.5.2 for $\mathsf{AC}^0[2]$ is the following algorithm:

> Given an n-variate Boolean function f, construct matrices $M_b = A_{a,b}(f)$ for $a = n/2 - \sqrt{n}$ and for every $0 \leq b \leq a$. Accept f if, for at least one b, $\mathrm{rank}(M_b) \geq \frac{2^n}{140n^2}$.

Natural Property useful against $\mathsf{AC}^0[p]$, for primes $p > 2$: Let f be a given n-variate Boolean function. Without loss of generality, assume n is odd. Denote by L the vector space of all multilinear polynomials of degree less than $n/2$ over $\mathbf{GF}(p)$. Let $\bar{f}$ be the unique multilinear polynomial over $\mathbf{GF}(p)$ that represents f on the Boolean cube $\{-1,1\}^n$ (after the linear transformation mapping the Boolean 0 to $1 \mod p$, and the Boolean 1 to $-1 \mod p$), i.e., f and $\bar{f}$ agree over all points of $\{-1,1\}^n$.

The natural property of Theorem 3.5.2 for $\mathsf{AC}^0[p]$ is the following algorithm:

> Given an n-variate Boolean function f, construct its unique multilinear polynomial extension $\bar{f}$ over $\mathbf{GF}(p)$. Accept f if $\dim(\bar{f}L + L) \geq \frac{3}{4} \cdot 2^n$ (over $\mathbf{GF}(p)$).

Theorem 3.5.2, in conjunction with Theorem 3.5.1, immediately yields our main application.

Corollary 28 (Learning $\mathsf{AC}^0[p]$ in quasipolytime)**.** *For every prime p, there is a randomized algorithm that, using membership queries, learns a given n-variate Boolean function $f \in \mathsf{AC}^0[p]$*

of size s_f to within error ε over the uniform distribution, in time $\mathsf{quasi\text{-}poly}(ns_f/\varepsilon)$.

Using Corollary 27, we also immediately get the following compression result, first proved (with somewhat stronger parameters) by Srinivasan [Sri15].

Corollary 29. *There is a randomized compression algorithm for depth-d* $\mathsf{AC}^0[p]$ *functions that compresses an n-variate function to the circuit size at most* $2^{n-n^{\mu}}$*, for* $\mu \geq \Omega(1/d)$.

3.5.3 Sketch of Complete Algorithm

Here, we sketch the algorithm implied by Theorem 3.5.1 for the case of $\mathsf{AC}^0[2]$. Let $f\colon \{0,1\}^n \to \{0,1\}$ be a function in $\mathsf{AC}^0[2]$ to be learned, given via membership oracle. Let R be a natural property, and let $L = n^{\mathsf{poly}(\log n)}$.

1. Design a subroutine for computing $\textsc{Amp}(f) = (f^k)^{GL}$ (Theorem 3.4.3) using f as an oracle.

2. Let D be a circuit simulating the natural property R_L. D is a distinguisher between $G_{\textsc{Amp}(f)}(s)$ for a random s and uniform, as shown in the proof of Theorem 3.5.1.

3. Convert D into C, a weak predictor for $\textsc{Amp}(f)$ on $(1/2+\Omega(1/L))$-fraction of inputs, using the NW reconstruction algorithm and oracle for $\textsc{Amp}(f)$ that we simulate using membership queries to f.

4. Use C as the oracle for the Goldreich-Levin reconstruction algorithm (Theorem 3.4.2), obtaining a predictor C' for the direct product of f.

5. Use C' as input to the Direct Product reconstruction algorithm of Theorem 3.4.1, and print the resulting circuit.

For the case of $\mathsf{AC}^0[p]$ with $p \neq 2$, the algorithm is essentially the same, but requires an additional step in the definition of $\textsc{Amp}(f)$: the von Neumann construction (Theorem 3.4.4) applied to $(f^k)^{GL}$. Thus, we need the von Neumann reconstruction step inserted between steps 3 and 4 of the complete algorithm above.

3.5.4 Natural properties useful against $\mathsf{AC}^0[p]$

For the remainder of this section, we present the natural properties useful against the class of AC^0 circuits with mod p gates, for any fixed prime p, as given in [RR97]. This material may be skipped at a first reading, as it is an exposition of previous results included here solely to make the presentation of our algorithm self-contained.

We follow the lower bound of Razborov [Raz87] (showing that MAJORITY is not in $\mathsf{AC}^0[2]$) to get a natural property useful against $\mathsf{AC}^0[2]$, and the lower bound of Smolensky [Smo87] (showing that PARITY is not in $\mathsf{AC}^0[p]$, for any prime $p \neq 2$) for the case of $\mathsf{AC}^0[p]$ for any prime $p > 2$. In both cases, the natural property is NC^2-computable, and is useful for circuit size up to $2^{\Omega(n^{1/(2d)})}$, where d is the circuit depth, and n is the input size.

The case of $\mathsf{AC}^0[2]$

Theorem 3.5.3 ([RR97])**.** *There is an NC^2-natural property of n-variate Boolean functions, with largeness at least $1/2$, that is useful against $\mathsf{AC}^0[2]$ circuits of depth d of size up to $2^{\Omega(n^{1/(2d)})}$.*

Proof. For $0 \leq a, b \leq n$, define a linear transformation $A_{a,b}$ that maps a Boolean function $f : \{0,1\}^n \to \{0,1\}$ to a matrix $M = A_{a,b}(f)$ of dimension $\binom{n}{a} \times \binom{n}{b}$, whose rows are indexed by size a subsets of $[n]$, and rows by size b subsets of $[n]$. For every $K \subseteq [n]$, define the set

$$Z(K) = \{(x_1, \ldots, x_n) \in \{0,1\}^n \mid \forall i \in K,\ x_i = 0\}.$$

For a size a subset $I \subseteq [n]$ and size b subset $J \subseteq [n]$, define

$$M_{I,J} = \oplus_{x \in Z(I \cup J)} f(x).$$

Razborov [Raz87] showed that if $\mathrm{rank}(A_{a,b}(f)) \geq \Omega(2^n/n^2)$, for $a = n/2 - \sqrt{n}$ and some $b \leq a$, then f requires $\mathsf{AC}^0[2]$ circuits of depth d of size at least $2^{\Omega(n^{1/(2d)})}$. He also showed the existence of an n-variate Boolean (symmetric) function h such that, for some $0 \leq b \leq a$,

$\text{rank}(A_{a,b}(h)) \geq \frac{2^n}{70n^2}$, and hence, h requires large $\mathsf{AC}^0[2]$ circuits.

This yields the following natural property useful against large $\mathsf{AC}^0[2]$ circuits:

> Given an n-variate Boolean function f, construct matrices $M_b = A_{a,b}(f)$ for $a = n/2 - \sqrt{n}$ and for every $0 \leq b \leq a$. Accept f if, for at least one b, $\text{rank}(M_b) \geq \frac{2^n}{140n^2}$.

First, observe that such a property of n-variate Boolean function f is computable in NC^2: we first construct $O(n)$ matrices of size at most $2^n \times 2^n$, and then compute the rank (over $\mathbf{GF}(2)$) of each of them. Thus, this property is NC^2-natural.

Secondly, by Razborov's result mentioned above, any f accepted by the property must require $\mathsf{AC}^0[2]$ depth d circuits of size at least $2^{\Omega(n^{1/(2d)})}$. Thus we have usefulness against exponential-size circuits.

Finally, to argue largeness, we use the function h mentioned above with $\text{rank}(A_{a,b}(h)) \geq \frac{2^n}{70n^2}$, for some $0 \leq b \leq a$. For each Boolean function f, we will show that either $A_{a,b}(f)$ or $A_{a,b}(f \oplus h)$ has rank at least $\frac{2^n}{140n^2}$, which implies that at least $1/2$ of all Boolean functions are accepted by our property.

Indeed, since $A_{a,b}$ is an $\mathbf{GF}(2)$-linear map, and using the subadditivity of rank, we get

$$\begin{aligned}\text{rank}(A_{a,b}(h)) &= \text{rank}(A_{a,b}(h \oplus f \oplus f)) \\ &\leq \text{rank}(A_{a,b}(h \oplus f)) + \text{rank}(A_{a,b}(f)).\end{aligned}$$

Thus, at least one of $A_{a,b}(f)$ or $A_{a,b}(f \oplus h)$ must have the rank at least $1/2$ of the rank of $A_{a,b}(h)$, as required. □

The case of $\mathsf{AC}^0[p]$ for all primes $p > 2$

Theorem 3.5.4 ([RR97])**.** *For every prime $p > 2$, there is an NC^2-natural property of n-variate Boolean functions, with largeness at least $1/2$, that is useful against $\mathsf{AC}^0[p]$ circuits of depth d of size up to $2^{\Omega(n^{1/(2d)})}$.*

Proof. Let f be a given n-variate Boolen function. Without loss of generality, assume n is odd.

Denote by L the vector space of all multilinear polynomials of degree less than $n/2$ over $\mathbf{GF}(p)$. Let $\bar{f}$ be the unique multilinear polynomial over $\mathbf{GF}(p)$ that represents f on the Boolean cube $\{-1,1\}^n$ (after the linear transfofrmation mapping the Boolean 0 to $1 \mod p$, and the Boolean 1 to $-1 \mod p$), i.e., f and $\bar{f}$ agree over all points of $\{-1,1\}^n$.

The natural property given by [RR97] is the following:

> Given an n-variate Boolean function f, construct its unique multilinear polynomial extension $\bar{f}$ over $\mathbf{GF}(p)$. Accept f if $\dim(\bar{f}L+L) \geq \frac{3}{4} \cdot 2^n$ (over $\mathbf{GF}(p)$).

It is easy to see that this property is computable in NC^2. It is also argued in [RR97] that this property has largeness at least $1/2$. Finally, it also follows from [RR97], based on Smolensky's lower bound proof [Smo87], that any n-variate function f accepted by this property must have $\mathsf{AC}^0[p]$ circuits of depth d of size at least $2^{\Omega(n^{1/(2d)})}$.

Indeed, Smolensky [Smo87] shows that, for every Boolean function f computed by an $\mathsf{AC}^0[p]$ circuit of depth d and size s, there exists a multilinear polynomial q over $\mathbf{GF}(p)$ of degree $D = O(\log^d(s/\varepsilon))$ that agrees with f on all but at most $w = \varepsilon 2^n$ points W of the Boolean cube, where we think of ε as a small constant (e.g., $\varepsilon = 0.2$). For any such f that also satisfies the condition $\dim(\bar{f}L+L) \geq \frac{3}{4} \cdot 2^n$, we will show that $D \geq \Omega(\sqrt{n})$. This would imply that any such f must have d-depth $\mathsf{AC}^0[p]$ circuits of size $2^{\Omega(n^{1/(2d)})}$, as required.

Suppose some f can be approximated by a multilinear degree D polynomial on all Boolean points except the set W of size $w \leq \varepsilon 2^n$, for small constant ε (to be determined). Suppose that f also satisfies the condition $\dim(\bar{f}L+L) \geq \frac{3}{4} \cdot 2^n$ over $\mathbf{GF}(p)$. Using Smolensky's arguments from [Smo87], we get that at least $p^{2^n(3/4-\varepsilon)}$ functions from $\{-1,1\}^n \setminus W$ to $\mathbf{GF}(p)$ are computable by multilinear polynomials over $\mathbf{GF}(p)$ of degree at most $(n-1)/2+D$. For $D = \lambda\sqrt{n}$, with some $\lambda > 0$, the number of distinct multilinear monomials of degree at most

$(n-1)/2+D$ is

$$\begin{aligned}
\sum_{i=0}^{(n-1)/2+D}\binom{n}{i} &= \sum_{i=0}^{(n-1)/2}\binom{n}{i} + \sum_{i=(n-1)/2+1}^{(n-1)/2+D}\binom{n}{i} \\
&\leq \frac{1}{2}\cdot 2^n + D\cdot\binom{n}{\frac{n-1}{2}} \\
&\leq \frac{1}{2}\cdot 2^n + D\cdot\frac{2^n}{\sqrt{\pi n/2}}\cdot\left(1+O\left(\frac{1}{\sqrt{n}}\right)\right) \qquad \text{(by Stirling's approximation)} \\
&\leq 2^n\cdot\left(\frac{1}{2}+\frac{2D}{\sqrt{\pi n}}\right) \\
&= 2^n\cdot\left(\frac{1}{2}+\lambda\cdot\frac{2}{\sqrt{\pi}}\right),
\end{aligned}$$

which can be made at most $(0.51)\cdot 2^n$, by taking λ a sufficiently small constant (e.g., $\lambda = \sqrt{\pi}/20$).

Thus, the number of distinct multilinear polynomials over $\mathbf{GF}(p)$ of degree at most $(n-1)/2+D$ is at most $p^{0.51\cdot 2^n}$. On the other hand, as mentioned above, there are at least $p^{(3/4-\varepsilon)2^n}$ functions from $\{-1,1\}^n \setminus W$ to $\mathbf{GF}(p)$ that are supposed to be computable by such low-degree multilinear polynomials. For ε small enough so that $3/4-\varepsilon > 0.51$ (e.g., $\varepsilon = 0.2$), we get that there are too many functions to be represented by low-degree polynomials. So it must be the case that the degree $D > \lambda\sqrt{n}$, for some constant $\lambda > 0$. □

3.6 NW designs cannot be computed in AC^0

In Section 3.3.1 we showed that NW designs (with parameters of interest to us) are computable by small $\mathsf{AC}^0[p]$ circuits, for any prime p. It is natural to ask if one can compute such NW designs by small AC^0 circuits, without modular gates. Here we show that this is *not* possible. Consider an NW design $S_1,\ldots,S_L \subseteq [n^2]$ where:

- each set S_i is of size n,
- the number of sets is $L = 2^\ell$ for $\ell = n^\varepsilon$ (for some $\varepsilon > 0$), and

- for any two distinct sets S_i and S_j, $i \neq j$, we have $|S_i \cap S_j| \leq \ell$.

To describe such a design, we use the following Boolean function g: for $1 \leq i \leq L$, and for $1 \leq k \leq n^2$, define $g(i,k) = 1$ iff $k \in S_i$. We will prove the following.

Theorem 3.6.1. *Let $g\colon \{0,1\}^{\ell+2\log n} \to \{0,1\}$ be the characteristic function for any NW design with the above parameters. Then g requires depth d AC^0 circuits of size $\exp(\ell^{1/d})$.*

To prove this result, we shall define a family of Boolean functions f_T, parameterized by sets $T \subseteq [n^2]$: for $1 \leq i \leq L$, we define $f_T(i) = 1$ iff $T \cap S_i \neq \emptyset$. Observe that if $g(i,k)$ is computable by AC^0 circuits of depth d and size s, then, for every set T, the function $f_T(i) = \vee_{k \in T}\ g(i,k)$ is computable by AC^0 circuits of depth at most $d+1$ and size $O(s \cdot |T|)$. Therefore, to prove Theorem 3.6.1, it will suffice to prove the following.

Lemma 3.6.2. *There exists a set $T \subseteq [n^2]$ such that $f_T\colon \{0,1\}^{\ell} \to \{0,1\}$ requires depth $d+1$ AC^0 circuits of size at least $\exp(\ell^{1/d})$.*

The idea of the proof of Lemma 3.6.2 is to show that for a random set T (of expected size $O(n)$), the function f_T has high average sensitivity (i.e., is likely to flip its value for many Hamming neighbors of a randomly chosen input). By averaging, we get the existence of a particular function f_T of high average sensitivity. On the other hand, it is well-known that AC^0 functions have low average sensitivity. This will imply that f_T must require large AC^0 circuits. We provide the details next.

Recall that the *sensitivity* of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ at input $x \in \{0,1\}^n$ is defined as the number of Hamming neighbors $y \in \{0,1\}^n$ of x (where y and x differ in exactly one coordinate i, $1 \leq i \leq n$) such that $f(x) \neq f(y)$. The *average sensitivity* of a function f, denoted $\mathbf{AS}(f)$, is the expected sensitivity of f at x, over uniformly random inputs $x \in \{0,1\}^n$. We use the following result by Boppana [Bop97].

Theorem 3.6.3 ([Bop97])**.** *The average sensitivity of a size s AC^0 circuit of depth d is at most $O((\log s)^{d-1})$.*

We shall prove the following simple claims that will imply that f_T has high average sensitivity, for a random $T \subseteq [n^2]$ of expected size $t = O(n)$. Below we shall choose a set $T \subseteq [n^2]$ by placing each index k, $1 \leq k \leq n^2$, into T with probability t/n^2, independently, for $t = n/2$. Clearly, the expected size of T is t.

Claim 30. *For every* $1 \leq i \leq L$, $\mathbf{Pr}_T\ [f_T(i) = 1] \approx 1/2$.

Proof. The probability a random set T misses all n positions of S_i is

$$\begin{aligned}\left(1 - \frac{t}{n^2}\right)^n &\approx 1 - \frac{tn}{n^2} \\ &= 1 - \frac{t}{n},\end{aligned}$$

which is approximately $1/2$ by our choice of t. □

Claim 31. *For every* $1 \leq i \neq j \leq L$, $\mathbf{Pr}_T\ [f_T(i) = 1 \wedge f_T(j) = 1] \leq 1/4 + o(1)$.

Proof. We have $\mathbf{Pr}_T\ [f_T(i) = 1 \wedge f_T(j) = 1]$ is equal to

$$\begin{aligned}&\mathbf{Pr}_T\ \left[T \cap (S_i \cap S_j) \neq \emptyset\right] \\ &\qquad + \mathbf{Pr}_T\ \left[f_T(i) = f_T(j) = 1 \mid T \cap (S_i \cap S_j) = \emptyset\right] \cdot \mathbf{Pr}_T\ \left[T \cap (S_i \cap S_j) = \emptyset\right]. \quad (3.8)\end{aligned}$$

Using the fact that $|S_i \cap S_j| \leq \ell = n^{\varepsilon}$ and arguing as in the proof of Claim 30, we get

$$\begin{aligned}\mathbf{Pr}_T\ \left[T \cap (S_i \cap S_j) \neq \emptyset\right] &\leq (t\ell)/n^2 \\ &= \ell/(2n) \\ &= o(1).\end{aligned}$$

Next we have $\mathbf{Pr}_T\ \left[f_T(i) = f_T(j) = 1 \mid T \cap (S_i \cap S_j) = \emptyset\right]$ equals

$$\mathbf{Pr}_T\ \left[f_T(i) = 1 \mid T \cap (S_i \cap S_j) = \emptyset\right] \cdot \mathbf{Pr}_T\ \left[f_T(j) = 1 \mid f_T(i) = 1 \wedge T \cap (S_i \cap S_j) = \emptyset\right].$$

Conditioned on T missing the intersection $S_i \cap S_j$, the conditional probability that T intersects S_i is

$$1 - \left(1 - \frac{t}{n^2}\right)^{n-|S_i \cap S_j|} \approx \frac{n - |S_i \cap S_j|}{2n} \leq \frac{1}{2}.$$

Similarly, conditioned on T missing the intersection $S_i \cap S_j$ but intersecting S_i, the conditional probability of T intersecting S_j is also approximately at most $1/2$ (following the same calculations as for the case of S_i above).

Putting everything together, we get that $\mathbf{Pr}_T\ [f_T(i) = 1 \wedge f_T(j) = 1] \leq 1/4 + o(1)$. □

Claim 32. *For every* $1 \leq i \neq j \leq L$, $\mathbf{Pr}_T\ [f_T(i) \neq f_T(j)] \geq \frac{1}{5}$.

Proof. For every fixed $i \neq j$, we have

$$\begin{aligned}\mathbf{Pr}_T\ [f_T(i) \neq f_T(j)] &\geq \mathbf{Pr}_T\ [f_T(i) = 1 \wedge f_T(j) = 0] \\ &= \mathbf{Pr}_T\ [f_T(i) = 1] - \mathbf{Pr}_T\ [f_T(i) = 1 \wedge f_T(j) = 1],\end{aligned}$$

which, by Claims 100 and 31, is at least $1/2 - 1/4 - o(1) = 1/4 - o(1) > 1/5$. □

Claim 33. *There exists a set* T *such that* $\mathbf{AS}(f_T) \geq \ell/5$.

Proof. For a string $x \in \{0,1\}^n$, we denote by $N(x)$ the set of all strings $y \in \{0,1\}^n$ that differ from x in exactly one coordinate; that is, $N(x)$ is the set of all Hamming neighbors of x in the Boolean cube $\{0,1\}^n$. Also, for a condition A, we denote by $\{A\}$ the indicator function of A, i.e., $\{A\} = 1$ if the condition A is true, and $\{A\} = 0$ otherwise.

We have

$$\begin{aligned}
\mathop{\mathbb{E}}_{T}\left[\mathbf{AS}(f_T)\right] &= \mathop{\mathbb{E}}_{T\subseteq[n^2],i\in\{0,1\}^\ell}\left[\sum_{j\in N(i)}\{f_T(i)\neq f_T(j)\}\right] \\
&= \mathop{\mathbb{E}}_{i}\left[\sum_{j\in N(i)}\mathop{\mathbb{E}}_{T}[\{f_T(i)\neq f_T(j)\}]\right] \\
&= \mathop{\mathbb{E}}_{i}\left[\sum_{j\in N(i)}\mathbf{Pr}_T[f_T(i)\neq f_T(j)]\right] \\
&\geq \mathop{\mathbb{E}}_{i}\left[\sum_{j\in N(i)}\frac{1}{5}\right] \qquad \text{(by Claim 32)} \\
&= \frac{\ell}{5}.
\end{aligned}$$

By averaging, there exists a set T, such that $\mathbf{AS}(f_T)\geq \ell/5$. □

Now we finish the proof of Lemma 3.6.2. Suppose the function f_T given by Claim 33 is computed by an AC^0 circuit of depth $d+1$ and size s. By Theorem 3.6.3, we get that $\mathbf{AS}(f_T)\leq O((\log s)^d)$. It follows that

$$\frac{\ell}{5}\leq O((\log s)^d),$$

which implies that $s\geq \exp(\ell^{1/d})$, as required. Thus, our techniques as currently constructed *cannot* learn AC^0 "directly" — by using the AC^0-natural property against AC^0.

3.7 Conclusions

For our applications, we need Λ strong enough to carry out a (version of) the construction, yet weak enough to have a natural property useful against it. Here we show that $\Lambda=\mathsf{AC}^0[p]$ for any prime p satisfies both conditions. A logical next step would be ACC^0: if one can get a natural property useful against ACC^0, for example by naturalizing Williams's [Wil14b] proof, then a learning algorithm for ACC^0 would follow. MOD_p can be simulated with MOD_m, $m=p\cdot a$ gates

by duplicating each input to the Mod_m gate a times (without any penalty in the number of gates), our construction for MOD_p can be applied directly by taking p to be any prime factor of m.

Connections between learning algorithms and lower bounds could also be explored in other settings. In particular, it would be interesting to give such a connection for arithmetic circuits. In [KI04], the NW generator is used to derandomize polynomial identity testing based on a polynomial with a large arithmetic circuit lower bound. Since the main reduction is constructive, one might hope to use it to design learning (or interpolation) algorithms for multivariate polynomials of small circuit complexity. However, it is unclear what the analogy of "natural property" would be in this setting.

We conclude with some open questions. Can we get an *exact* compression algorithm for $\mathsf{AC}^0[p]$ (or even AC^0) functions that would produce circuits of *subexponential* size? Can our learning algorithm be derandomized? Is there a way to get nontrivial SAT algorithms from natural properties? This is a question about the *fine-grained* NP-hardness of MCSP. Finally, are there more applications of a "play-to-lose" attitude towards pseudorandom constructions?

Chapter 3, in part, is based on the material as it appears in "Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Learning algorithms from natural proofs. In Ran Raz, editor, *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, volume 50 of *LIPIcs*, pages 10:1–10:24. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016". The dissertation author was the primary investigator and author of this paper.

Chapter 4

Agnostic Natural Learning

4.1 Introduction

Recently many new connections have been discovered between two complementary domains: proving circuit lower bounds and designing meta-computational algorithms for the corresponding circuit classes (see, e.g., [San10, Wil10, Wil11, IMP12, IMZ12, CKK$^+$15, CIKK16]). In particular, [CIKK16] shows that a natural property (in the sense of Razborov and Rudich [RR97]) for a (sufficiently powerful) circuit class Λ yields an efficient PAC learning algorithm for the same circuit class, under the uniform distribution, with membership queries; this approach led to a first learning algorithm for the class $\mathsf{AC}^0[q]$ (of constant-depth circuits with AND, OR, NOT, and modulo q gates), for every prime q, covered by the previous chapter of this thesis.

Here, we extend those results. The "learning algorithms from natural proofs" technique of [CIKK16] applies only to *realizable case* learning: if a function f is computed exactly by an appropriate circuit class Λ for which there is a natural proof of a circuit lower bound, then we can learn f using membership queries in time dependent on the strength of the circuit lower bound. A more realistic learning model is *agnostic learning*, where we select some "touchstone" class Λ and attempt to find a hypothesis that isn't "too far off" from the best Λ-approximation to the target function.

We show that, even in this agnostic setting, we can (somewhat generically) obtain learning

algorithms from natural proofs. We instantiate this framework to give the first membership-query agnostic learning algorithm over the uniform distribution for $\mathsf{AC}^0[q]$, the class of constant-depth circuits of polynomial-size with unbounded fanin AND, OR, NOT, and MOD_q gates. Previously, only the case of AC^0 circuits was known — albeit for an agnostic algorithm without membership queries, and with better approximation error [KKMS08].

Theorem 34 ($\mathsf{AC}^0[q]$ agnostic learning)**.** *Let q be any prime. There is a randomized quasi-polynomial-time algorithm such that, given oracle access to a function $f\colon \{0,1\}^n \to \{0,1\}$ that agrees with some unknown function in $\mathsf{AC}^0[q]$ on at least $1-\beta$ fraction of inputs (for some non-negligible $\beta > 0$), the algorithm outputs a circuit that computes f on all but at most $\mathsf{poly}(\log n)\cdot\beta$ fraction of inputs.*

As an interesting special case, we get a quasipolynomial-time agnostic learning algorithm for n-variate polynomials over $\mathrm{GF}(q)$ of low degree (say, at most $\mathsf{poly}(\log n)$), for prime $q \geq 2$ (as every polynomial of degree d is computable by an $\mathsf{AC}^0[q]$ circuit of size $O(n^d)$). Before our result, no such learning algorithm for polynomials was known.

For an algorithm with error $c(n)\cdot\beta$, for some function c, we call the factor $c(n)$ the *weakness* parameter of the learning algorithm. It is desirable to have $c(n) = 1$. Our algorithm for $\mathsf{AC}^0[q]$ above has weakness $\mathsf{poly}(\log n)$. In general, we have a trade-off between the quality of a natural property for the circuit class, and the quality of the resulting agnostic learning algorithm for the same class. For simplicity, we state here just the result for the best-case scenario; see Theorem 48 on page 97 for the fully general statement.

Theorem 35 (Ideal-case trade-off)**.** *Suppose there is a natural property for a circuit class $\mathscr{C} \supseteq \mathsf{AC}^0[2]$ that is useful against functions that agree on $1/2 + \exp(-\Omega(n))$ of inputs with some function of $\mathscr{C}$-circuit complexity $\exp(\Omega(n))$. Then, for some constant $c > 0$, there is a polynomial-time query agnostic learning algorithm for $\mathscr{C}$ with weakness c.*

Theorem 35 yields a "search-to-decision" reduction for a version of the Minimal Circuit Size Problem (MCSP). Define the Minimal Approximate Circuit Size Problem (MACSP) as

follows: Given a truth table of an n-variate boolean function f, and parameters $s \in \mathbb{N}$ and $\delta \in [0,1]$, decide if there exists a boolean circuit C of size at most s that agrees with f on all but at most δ fraction of inputs. (MCSP is a special case of MACSP for $\delta = 0$.) Clearly, if MACSP is easy (say, in P), then, for a given size bound s (our "budget"), we can determine the best approximation parameter δ for every given truth table of a boolean function f. But, since MACSP is an ideal-case tolerant natural property for general circuits, we get by Theorem 35 that a polynomial-time algorithm for MACSP would yield a polynomial-time algorithm to actually find a circuit of size $\mathsf{poly}(s)$, with an approximation guarantee $O(\delta)$.[1]

Another way to interpret Theorem 35 is as follows. If MACSP is in P, then, given oracle access to a boolean function f, and a budget $s \in \mathbb{N}$, we can learn, in polynomial time, a circuit of size $\mathsf{poly}(s)$ that agrees with f on all but at most $O(\delta)$ fraction of inputs, where δ is the error of the best size s circuit for f. That is, we can learn essentially the best possible circuit for f, given our budget s on the circuit size.

4.1.1 Our approach

The key observation in adapting to the agnostic setting is that many natural properties contain even more useful distinguishers than required for realizable-case learning. As defined by [RR97], the distinguisher from a natural property rejects truth tables that are exactly computed by Λ-circuits. But existing natural properties give us something even stronger: they reject truth tables which are just *close* to those computed by Λ-circuits. Using this observation and the same "play to lose" distinguisher-to-predictor reduction as in [CIKK16], we obtain agnostic learning algorithms from such natural properties.

More precisely, we show that if a natural property for a circuit class Λ (containing $\mathsf{AC}^0[q]$) is *tolerant* in the sense that it distinguishes from random the truth tables of functions "close" to

[1] In [CIKK16], a similar "search-to-decision" reduction was given for MCSP: if a given boolean function f is *exactly* computable by a polynomial-size circuit, then one can find a polynomial-size circuit *approximately* computing f, given a polynomial-time algorithm for MCSP. In contrast, here we say that if f can be non-trivially *approximated* by a polynomial-size circuit, we can find another polynomial-size circuit that achieves the *same approximation error* up to a constant factor, given a polynomial-time algorithm for MACSP.

the class Λ (of "large" circuit complexity), then it can be used to get an agnostic membership-query algorithm for learning Λ. We argue that such a tolerant natural property exists for $\mathsf{AC}^0[q]$ [Raz87, Smo87, RR97], which is then used to prove our Theorem 34. For $\mathsf{AC}^0[2]$, we need to dig inside the arguments of [Raz87], and show that his original circuit lower bound proof does yield a certain tolerant natural property. For $\mathsf{AC}^0[q]$, for prime $q > 2$, we actually need to re-do the "natural proof" argument of [RR97] by adapting it to the case of $\mathrm{GF}(q)$-valued functions (rather than boolean functions). Not only does it allow us to get tolerant natural properties for $\mathsf{AC}^0[q]$, but also simplifies and streamlines the analysis in [CIKK16] of the learning algorithm for $\mathsf{AC}^0[q]$.

By definition, tolerant natural properties can be used for proving *average-case* circuit lower bounds (as opposed to the worst-case circuit lower bounds implied by standard natural properties). Thus the main message of the present chapter can be summarized as follows:

> Natural proofs of *average-case* circuit lower bounds imply *agnostic* learning algorithms!

In contrast, the main result of [CIKK16] says that natural proofs of worst-case circuit lower bounds imply standard (non-agnostic) learning algorithms.

4.1.2 Our techniques

We build upon the framework of [CIKK16] who use a natural property for a given circuit class Λ in order to devise a learning algorithm for the same class. Recall that a natural property (in the sense of [RR97]) is an efficient algorithm that tells apart truth tables of functions in the class Λ (of some "large" circuit complexity u) from those of random functions. To learn a function $f \in \Lambda$, for some circuit class Λ that has an associated natural property, the idea is to apply (as only a thought experiment!) an appropriate "function generator" that maps f to a family of functions all of which are "easy" (of small Λ circuit complexity) and so will be rejected by the natural property for the class. Thus an efficient algorithm from the natural property acts as a distinguisher "breaking" the function generator. If the function generator has an "efficient

reconstruction" property, meaning that a distinguisher for the generator can be used to build a small circuit approximately computing the original function f, we get a learning algorithm for f. Thus, the actual learning algorithm is using the natural property algorithm as a distinguisher, and applies the efficient reconstruction procedure (associated with the given function generator) to build a circuit approximating f. Usually, such a reconstruction procedure requires oracle access to the function generator; if, however, the function generator is "local" in the sense that such oracle access to the generator can be efficiently reduced to oracle access to the original function f, one gets a query learning algorithm for the concept class Λ.

To adapt this approach to the case of agnostic learning, where a function f to be learned is not in the class Λ, but rather just somewhat close to the class, we need to satisfy the following requirements: (1) the outputs of the function generator applied to f must be close to the class Λ (of some circuit size u), and (2) the natural property for Λ must reject not only functions in Λ (of size u), but also functions that are close to those.

We call natural properties satisfying condition (2) above *tolerant*. We say that a natural property has ρ-tolerant u-usefulness for the circuit class Λ if it rejects all truth tables of functions that agree with some function in $\Lambda[u]$ (computable by a Λ circuit of size u) on all but at most ρ fraction of inputs. We show that the natural property for the circuit class $\mathsf{AC}^0[2]$ from [Raz87] is in fact ρ-tolerant, for some small but nontrivial $\rho > 0$, and with large (weakly-exponential) usefulness u.

With tolerant natural properties in hand, we turn to requirement (1) above: getting the truth tables output by the function generator on a given function f to be close to those from the circuit class $\Lambda[u]$. We need to take a closer look at the function generator used in [CIKK16]. It comprises two components: (1) amplification, and (2) Nisan-Wigderson (NW) generator [NW94] applied to the amplified version $\mathsf{Amp}(f)$ of the function f. The purpose of the amplification component is to "error-correct" f so that even a circuit that computes $\mathsf{Amp}(f)$ with small advantage over random guessing can be used to construct a circuit that computes f almost everywhere. The NW generator applied to $\mathsf{Amp}(f)$ has the properties required of the

function generator: locality and efficient reconstruction.

In our case, suppose that f agrees with some function $h \in \Lambda$ on a large fraction of inputs. Once we apply amplification to both f and h, we get $\mathsf{Amp}(f)$ and $\mathsf{Amp}(h)$ that are pushed further apart (as one would expect when using error-correcting codes). In order to keep the amplified functions close to each other, we will tone down the amplification procedure, which will adversely affect the approximation error of our learning algorithm, but the error can still be kept relatively small.

Next we need to ensure that the NW generator when applied to $\mathsf{Amp}(f)$ generates a family of functions such that most of them are sufficiently close to the family generated on $\mathsf{Amp}(h)$. In other words, we would like the generator to almost preserve the relative distance between the functions it is applied to. This can be achieved as follows. First, we observe that the definition of the NW generator guarantees that on a random seed z, the functions generated for $\mathsf{Amp}(f)$ and $\mathsf{Amp}(h)$ have the expected distance (over random z) equal to the actual distance between $\mathsf{Amp}(f)$ and $\mathsf{Amp}(h)$. Thus we have distance preservation in expectation. To make it concentrated around the expectation, we modify the NW construction by adding a pairwise-independent generator inside the NW construction. This ensures that the truth tables output by the modified NW generator are evaluations of $\mathsf{Amp}(f)$ (or $\mathsf{Amp}(h)$) on a sequence of pairwise independent inputs. The required concentration then follows by the Chebyshev bound. A similar modification of the NW generator was done in [IW97], where an expander-walk generator was used for even better concentration; we use a simple pairwise generator as it can be easily implemented in $\mathsf{AC}^0[2]$, and it provides sufficient concentration for our purposes.

4.1.3 Related work

The concept of *agnostic* learning was introduced by Kearns et al. [KSS94], where it was also shown that piecewise linear functions are agnostically learnable. Agnostic learning is also known for certain geometric patterns [GKS95], and restricted neural networks [LBW96]. More results are known for the restricted versions of agnostic learning, for instance, when the distribu-

tion over examples is uniform. The class of AC^0 functions was shown to be (weakly) agnostically learnable under the uniform distribution by [KSS94]. It was later shown by [KKMS08] that the well-known LMN learning algorithm of [LMN93] achieves a constant-factor approximation of the optimal error (improved to the constant factor 2 in [Jac06]), and that a modification of the algorithm (using L_1 regression) achieves the optimal error; the runtime of the algorithm is quasipolynomial. In fact, the result of [KKMS08] is generic in the following sense: any concept class of functions with certain "Fourier concentration" (as is the case, e.g., for AC^0 functions by the results of [LMN93]) admits an agnostic learning algorithm under the uniform distribution, with an optimal error, whose runtime depends on the strength of the Fourier concentration for the concept class.

In distribution-independent setting, allowing membership queries does not give extra power to agnostic learning, yet membership queries can help when the distribution is uniform [Fel09]. In particular, under the uniform distribution, Gopalan, Kalai and Klivans [GKK08] and Feldman [Fel10] give polynomial-time agnostic learning algorithms with membership queries for decision trees.

Agnostic learning of parities is closely related to the well-studied problem of learning noisy parities, which has a number of applications beyond learning theory, from decoding random linear codes to cryptography [BFKL93, FS96, Ale03, Lyu05, Pie12].

Under the uniform distribution, agnostic learning of parities (that is, learning parities with adversarial noise) reduces to learning parities with random noise [FGKP06]. Blum, Kalai and Wasserman [BKW03] give an algorithm that properly learns length k parities with random noise under uniform distribution in time and sample size $\mathsf{poly}((1/(1-2\eta))^{2^a}, 2^b)$, where $\eta < 1/2$ is the noise probability, and $ab \geq k$. This is in contrast to the NP-hardness of properly learning noisy parities under arbitrary distributions, which follows from [Hås01]. Later, Lyubashevsky [Lyu05] improved query complexity of the [BKW03] algorithm to $n^{1+\varepsilon}$, at the expense of bringing the running time up to $2^{O(n/\log\log n)}$, for $\eta < 1/2 - 2^{-(\log n)^\delta}$ for a constant δ. A corollary of the latter result is a subexponential algorithm for decoding $n \times n^{1+\varepsilon}$ random binary linear codes, in

the random noise setting.

Regev [Reg09] considered an extension of learning parity with noise to mod p, which he called LWE (learning with error). He has shown that an efficient solution to LWE (for some range of parameters) implies an efficient quantum approximation of two variants of the shortest vector problem (GapSVP and the shortest independent vectors problem) and presented a public-key cryptosystem based on its hardness.

Remainder of the chapter.

We start with some basic definitions in Section 4.2. In Section 4.3, we prove our main result, Theorem 34, by instantiating the "agnostic learning from tolerant natural properties" framework to the case of $\mathsf{AC}^0[q]$ circuits, for any prime q. We present this framework in full generality in Section 4.4, where, in particular, we prove Theorem 35. In Section 4.5, we discuss the difficulty of removing membership queries from our agnostic learning algorithms for $\mathsf{AC}^0[2]$ (as it would have consequences for learning noisy parities). We conclude with some open questions in Section 4.6.

4.2 Preliminaries

For n-variate boolean functions f and g, we define the distance between them, denoted $\mathrm{DIST}(f,g)$, to be the number of inputs x where $f(x) \neq g(x)$. We denote by $\mathrm{dist}(f,g)$ the relative distance $\mathrm{DIST}(f,g)/2^n$. For a class $\mathscr{F}$ of n-variate boolean functions, and an n-variate boolean function f, we define the distance of f from the class $\mathscr{F}$, denoted $\mathrm{DIST}(f,\mathscr{F})$, as $\min_{h\in\mathscr{F}} \mathrm{DIST}(f,h)$. The relative distance of f from $\mathscr{F}$ is $\mathrm{dist}(f,\mathscr{F}) = \mathrm{DIST}(f,\mathscr{F})/2^n$.

4.2.1 Learning algorithms

The concept of *agnostic learning* was introduced by [KSS94]. As in the PAC model of Valiant [Val84], we have a distribution over labeled examples $(x, f(x))$ for some function f, and we wish to learn f up to a small additive error over the given distribution. However, unlike in

the PAC model, we don't assume that f belongs to some concept class $\mathscr{C}$, but rather that f is "close" to $\mathscr{C}$. More precisely, setting opt to be the disagreement probability between f and the best (closest) function $h \in \mathscr{C}$, the agnostic learning algorithm is supposed to output, with high probability $1-\delta$, a hypothesis that disagrees with f with probability at most $\mathsf{opt}+\varepsilon$, for given $\varepsilon, \delta \in [0,1]$. If the underlying distribution over examples is uniform, we say that the concept class $\mathscr{C}$ is agnostically learnable under the uniform distribution.

In the special case where we allow membership oracle, i.e., our learning algorithm has oracle access to the function f it is trying to learn, we call it a (membership) query agnostic learning algorithm. If, in addition, the hypothesis error is measured under the uniform distribution, we call it a query agnostic learning algorithm under the uniform distribution.

The learning algorithms considered in this chapter are query algorithms under the uniform distribution. However, they don't achieve the ideal error $\mathsf{opt}+\varepsilon$. Rather, we get an error of the form $c(n) \cdot \mathsf{opt}$, for some function c, which we call the *weakness* parameter of the agnostic learning algorithm; we also assume that opt is non-negligible and so we can drop the additive error ε to simplify the notation. For example, in the case of $\mathscr{C} = \mathsf{AC}^0[2]$, our learning algorithm has weakness $\mathsf{poly}(\log n)$.

4.2.2 Tolerant natural properties

We extend the definition of a natural property [RR97] to the case of a *tolerant* property, which intuitively says that not only all "easy" functions are rejected by the property, but also all functions "sufficiently close" to the "easy ones" are rejected. Such tolerant properties yield not just worst-case, but also average-case circuit lower bounds.

Let F_n be the collection of all Boolean functions on n variables. Λ and Γ denote complexity classes. A combinatorial property is a sequence of subsets of F_n for each n.

Definition 36 (Tolerant Natural Property)**.** A combinatorial property $\{R_n\}_{n\geq 0}$ is Γ-natural with density δ and τ-tolerant u-usefulness, for some functions $\delta, \tau : \mathbb{N} \to [0,1]$ and $u : \mathbb{N} \to \mathbb{N}$, if it satisfies the following conditions:

Γ-Constructivity: Given the truth table of f_n, a Γ-algorithm decides if $f_n \in R_n$.

δ-Largeness: $|R_n| \geq \delta(n) \cdot |F_n|$.

τ-Tolerant u-Usefulness: For all $f_n \in F_n$ (for large n), if $\text{dist}(f_n, \Lambda[u(n)]) \leq \tau(n)$, then $f_n \notin R_n$.

The standard natural property [RR97] is 0-tolerant in our language. For a number of complexity classes, including $\mathsf{AC}^0[q]$ for primes q, 0-tolerant natural properties were given in [RR97]. We prove that the natural property of [Raz87] has in fact $(1/n^3)$-tolerant usefulness against d-depth $\mathsf{AC}^0[2]$ circuits of size $\exp(\Omega(n^{1/(2d)}))$.

Lemma 37 (Tolerant natural property for $\mathsf{AC}^0[2]$)**.** *There is a* P*-natural property* $\{R_n\}_{n\geq 0}$ *with largeness* $1/2$*, and* $(1/n^3)$*-tolerant* $\exp(\Omega(n^{1/(2d)}))$*-usefulness against* d*-depth* $\mathsf{AC}^0[2]$ *circuits.*

We also give a tolerant natural property against $\mathsf{AC}^0[q]$ for prime q.

Lemma 38 (Tolerant natural property for $\mathsf{AC}^0[q]$)**.** *There is a* P*-natural property* $\{R_n\}_{n\geq 0}$ *with largeness* $1/2$*, and* $(.15)$*-tolerant* $\exp(\Omega(n^{1/(2d)}))$*-usefulness against* d*-depth* $\mathsf{AC}^0[q]$ *circuits computing functions* $f\colon \{1,-1\}^n \to \text{GF}(q)$.

We defer the construction of these natural properties to Section 4.3.6, after using them in our applications.

4.3 Agnostic learning from tolerant natural properties for $\mathsf{AC}^0[2]$

4.3.1 The CIKK framework

Recall the way non-agnostic learning algorithms follow from natural properties in the framework of [CIKK16]. Suppose we want to learn a function f in some circuit class Λ; for simplicity, assume f has polynomial-size circuits of type Λ.

As a thought experiment, imagine the following transformations applied to f. First, we *amplify* f, getting a new function $F = \textsc{Amp}(f)$, on polynomially larger inputs, with the property:

> if we are given a small circuit computing F on at least $1/2+\varepsilon$ fraction of inputs, then we can construct a circuit computing the original function f on at least $1-1/\mathsf{poly}(n)$ fraction of inputs, in randomized time $\mathsf{poly}(n,1/\varepsilon)$, using membership queries to f.

Then F is used as a "hard function" for the NW generator G. For each seed z of the NW generator, we view the output binary string $G(z)$ of length L as the truth table of an ℓ-variate boolean function, for $\ell = \log L$. The crucial observation in [CIKK16] is that the circuit complexity of this ℓ-variate boolean function is polynomial in the circuit size of the original function f, which is $\mathsf{poly}(n)$.

We need to express this circuit complexity $\mathsf{poly}(n)$ as a function of the input size ℓ. Note that if the stretch L is small, for example, if $L = \mathsf{poly}(n)$, then $\ell = O(\log n)$, and so the ℓ-variate function (whose truth table is) output by $G(z)$ has circuit complexity exponential in its input size ℓ. Thus, to reduce the circuit complexity of the function output by $G(z)$, we need to increase the stretch L of the NW generator. For example, by taking $L = \exp(\mathsf{poly}\log n)$, we can ensure that the circuit complexity of $G(z)$ (for each seed z) is only weakly exponential in the input size ℓ.

The point of using the NW generator to produce truth tables of relatively easy functions $G(z)$ is that we assumed the existence of an efficient natural property (with sufficient usefulness) which will accept many random truth tables, but will reject all truth tables of easy functions. In other words, this natural property provides an efficient (polynomial-time) algorithm that *distinguishes* the outputs of the NW generator G from truly random strings. But then, the analysis of the NW generator construction implies that we get from this distinguisher a new algorithm that computes F (the function upon which the NW generator was based) on at least $1/2+\Omega(1/L)$ fraction of inputs; where the reconstruction algorithm requires membership oracle for f. The latter implies — by the aforementioned properties of $F = \textsc{Amp}(f)$ — that we can construct a circuit computing f on almost all inputs, in time $\mathsf{poly}(n,L)$ (again, using membership oracle for f). Thus we get a learning algorithm from the natural property, using the efficient reconstruction algorithm for the NW generator and the amplification procedure.

For example, using natural properties against $\mathsf{AC}^0[2]$ that are useful against circuits

of weakly-exponential size [RR97], the above framework yields a learning algorithm, with membership queries, for functions computable by polynomial-size $\mathsf{AC}^0[2]$ circuits, running in quasipolynomial time.

4.3.2 Extension to the agnostic learning case

We wish to apply the same framework to the task of agnostic learning. Suppose we wish to learn a function f which is only somewhat close to a function h in some circuit class Λ (of polynomial-size circuits). Suppose that $\mathrm{dist}(f,\Lambda) \leq \beta$, and that $h \in \Lambda$ is the closest function to f. Assume we are given a membership oracle for f.

To apply the [CIKK16] approach to learn f, we need to ensure the following:

> for *most* seeds z, the function $G(z)$ (for the NW generator based on $F = \textsc{Amp}(f)$) is rejected by the appropriate natural property for our circuit class Λ.

If so, then we have a distinguisher for the NW generator based on F, and, as before, can efficiently construct a circuit for computing f almost everywhere.

As f is not in the class Λ, but rather just close to it, the best we can hope for is that the amplified function $F = \textsc{Amp}(f)$ is also somewhat close to Λ, and that the outputs of the NW generator $G(z)$ based on F are also somewhat close to the class Λ (of larger circuit size). If we can guarantee that (most of) the strings $G(z)$ are at relative distance at most τ from $\Lambda[u]$, then our natural property with τ-tolerant u-usefulness will be a distinguisher for the NW generator, and we can reconstruct a circuit approximately computing f.

We need to balance the opposing constraints. On the one hand, to keep $F = \textsc{Amp}(f)$ close to Λ, we cannot amplify f too much, as the amplification, like an error-correcting encoding, pushes the originally close functions far apart. On the other hand, the stronger the amplification applied to f, the smaller the approximation error we get from a circuit for f constructed by the learning algorithm. As we are restricted by the tolerance parameter τ of our natural property, we are forced to keep the amplification relatively weak, which in turn implies a weak approximation error for the learned circuit for f.

Suppose that $f\colon \{0,1\}^n \to \{0,1\}$ is at the relative distance β from some n-variate function $h \in \Lambda[\mathsf{poly}]$. We will fine-tune the amplification procedure of [CIKK16] so that $F = \textsc{Amp}(f)$ and $H = \textsc{Amp}(h)$ are at the relative distance at most $\mu(n)$, for some $\mu : \mathbb{N} \to [0,1]$ to be determined. Then we need to ensure that the outputs of the NW generator on F and on H, for most random seeds z, produce truth tables of length L that are at the relative distance at most $\tau(\ell)$ from each other, where $\ell = \log L$ is the input size of such a function output by $G(z)$.

To ensure that the NW generator based on close functions F and H produces strings that are close (for most seeds z), we modify the NW generator by adding a pairwise-independent generator as an extra component. (Similar modification to the NW generator, using an expander-walk generator, was done in [IW97], for a different purpose.) We will show that such a modified NW generator, when run on functions F and H that are at the relative distance $\mu(n)$ from each other, indeed outputs, for most seeds z, strings $G^F(z)$ and $G^H(z)$ of length L each, which are at the relative distance at most $2\mu(n)$ from each other. Expressing $2\mu(n)$ as a function of the input length $\ell = \log L$, we get an upper bound on the relative distance between $G^F(z)$ and $\Lambda[u]$ (as $G^H(z) \in \Lambda[u]$ by our assumption that $h \in \Lambda[\mathsf{poly}]$), for most seeds z. Here we choose the stretch L long enough so that the circuit complexity of the functions $G^H(z)$ is at most u, where u is usefulness of our natural property. For example, for $\mathsf{AC}^0[2]$, we have usefulness against weakly-exponential circuit size $\exp(n^{1/(2d)})$ for depth d circuits, and so we can make L to be quasi-polynomial, $\exp(\mathsf{poly}\log n)$.

4.3.3 Outline of the general method

In converting a tolerant natural property to an agnostic learning algorithm, we go through the following steps, mostly analogous to the steps in [CIKK16].

Initial assumptions We start with access via membership queries to a Boolean function f. We are promised that there is a function $h \in C$ so that $\mathrm{dist}(f,h) \leq \beta$, for some parameter β. We do not have any access to h, but can refer to it in the analysis.

Amplification The first step is to perform an *amplification construction*, $\mathsf{Amp}(f)$, to obtain a function F. Similarly, we can (conceptually) apply $\mathsf{Amp}(h)$ to obtain a function H. We need the following properties:

1. We can simulate membership queries to F via membership queries to f
2. $H \in C$
3. We can bound $\mathrm{dist}(F,H)$ away from $1/2$. The exact bound we will require will depend on the tolerance of the natural property.

Pseudo-random Function Generator We next convert F to a *pseudo-random function generator*, $G_s^F(I)$ (and, conceptually, convert H into $G_s^H(I)$. For each seed s, G_s^F is a Boolean function on ℓ bits, producing a truth table of size $L = 2^\ell$. We call L the *stretch* of the generator. We need the following properties:

1. Given s, the truth table for G_s^F can be computed via membership queries to F (and hence, f).
2. For each s, $G_s^H(I)$ has small C circuit complexity (as a function of ℓ bit input I)
3. With good probability over s, $\mathrm{dist}(G_s^F, G_s^H)$ is small

Again, the exact quantitative requirements will depend on the quality of the tolerant natural property. The stronger the circuit lower bound the property is useful against, the smaller we can make the stretch and so the larger the relative circuit complexity of G_s^H in (2) can be. The more tolerant the property is, the larger the allowed distance in (3) can be. The greater the density, the smaller the probability over seeds of small distance between G_s^F and G_s^H in (3) can be.

Apply tolerant natural property to get a distinguisher Now we use the tolerant natural property as a *distinguisher*, telling the difference between G_s^F and a random function of the same size. The second and third conditions above imply that, for many seeds s, G_s^F is

close to a function with small C complexity. Thus, the property will not hold for many such functions (as long as close is within the tolerance, and small within the usefulness of the property). On the other hand, largeness implies that it will hold for many random functions. A gap between these two probabilities implies a distinguishing probability. The size of the distinguisher we obtain will depend on the stretch L and the constructivity of the property.

Convert distinguisher to a predictor We use the contrapositive of the correctness proof of the PRFG construction to obtain a *predictor* that non-trivially predicts F. Note that non-trivially usually means with advantage at most $1/L$ over random guessing, so the smaller the stretch, the better the predictor will be.

Reverse the amplification Finally, we apply the converse of the hardness amplification correctness proof to obtain a circuit that computes the original function f with good probability. Note that the agreement of the circuit for f will depend on the strength of the hardness amplifier we can use (which is largely determined by the tolerance) but also on the prediction advantage (largely determined by the stretch, itself determined by the usefulness of the property). Thus, the strongest results will only apply when the tolerance is exponentially close to $1/2$ and the usefulness is exponential.

4.3.4 The case of $\mathsf{AC}^0[2]$

We first consider the case of amplification for $\mathsf{AC}^0[2]$. The case of $\mathsf{AC}^0[q]$ for primes $q > 2$ can be done in a similar way, where we work with $\mathrm{GF}(q)$-valued rather than Boolean functions; we sketch the argument in Section 4.3.5 below.

Given a boolean function $f\colon \{0,1\}^n \to \{0,1\}$, and a parameter $k = k(n) \in \mathbb{N}$, the amplification $\mathsf{Amp}_k(f)$ is defined as the Goldreich-Levin (Hadamard code) encoding of the k-wise

direct product of f:

$$\textsc{Amp}_k(f) = F(x_1,\ldots,x_k,b_1,\ldots,b_k) = \sum_{i=1}^{k} b_i \cdot f(x_i),$$

where $x_1,\ldots,x_k \in \{0,1\}^n$, $b_1,\ldots,b_k \in \{0,1\}$, and the summation is modulo 2.

It is shown in [CIKK16] that the error parameter of the learning algorithm for f is a function of k and the stretch L of the generator.

Theorem 39 ([CIKK16])**.** *Suppose the NW generator based on the function $F = \textsc{Amp}_k(f)$, with output strings of length L, is broken with a constant distinguishing probability. Then, using the distinguisher and membership queries to f, one can construct a circuit computing f on at least $1-\varepsilon$ fraction of inputs, for $\varepsilon \leq O((\ln L)/k)$. The construction algorithm is a randomized $\mathsf{poly}(n,k,L)$-time algorithm.*

Suppose there is a function $h \in \mathsf{AC}^0[2]$ such that $\mathrm{dist}(f,h) = \beta$. As observed in [CIKK16], the function $H = \mathsf{Amp}_k(h) \in \mathsf{AC}^0[2]$ for any $k = k(n) \leq \mathsf{poly}(n)$. It is also easy to argue that $\mathrm{dist}(F,H) = 1/2 - (1-\beta)^k/2$. For a given $\tau = \tau(\ell)$, we want to choose k so that $\mathrm{dist}(F,H) \leq \tau/4$. That is, we want $(1-\beta)^k \geq 1-\tau/2$. Using the inequalities $1+x \leq e^x$ (true for all x), and $1-x \geq e^{-2x}$ (true for all $0 \leq x \leq 0.7$), we are allowed to take $k = \tau/(4\beta)$.

Then the NW generator based on F outputs a truth table of an ℓ-variate function that has the expected (over random seeds z to the generator) relative distance at most $\tau/4$ from the class of $\mathsf{AC}^0[2]$ circuits of size u, for weakly-exponential circuit size u (for which we have a tolerant natural property given by Theorem 37). By Markov's inequality, we get that the actual distance is at most τ for at least $3/4$ fraction of the random seeds z to the generator.[2] Thus, for $\mathsf{AC}^0[2]$, we can make the stretch L of our generator to be quasipolynomial, $L = \exp(\mathsf{poly}(\log n))$. Then $\ell = \log L = \mathsf{poly}(\log n)$.

[2]Here, and for the case of $\mathsf{AC}^0[q]$ for primes $q > 2$ later, we can use a simple averaging argument and keep the NW generator as is, because we have natural properties for these classes with very poor tolerance parameters. However, for the general case, when we may have better tolerance parameters, we achieve better concentration by combining the NW generator with a pairwise-independent generator.

As we have $(1/\ell^3)$-tolerant natural property for $\mathsf{AC}^0[2]$ circuits of size u computing ℓ-input boolean functions (Theorem 37), we set $\tau = (1/\ell^3)$, and get that $k = (4\beta\ell^3)^{-1}$. As the τ-tolerant natural property breaks the NW generator based on F, we get by Theorem 39 that f can be learned up to the error $O((\log L)/k) \leq O(\beta \cdot \ell^4) \leq \mathsf{poly}(\log n) \cdot \beta$.

Thus we have proved the following.

Theorem 40 (Agnostic learning of $\mathsf{AC}^0[2]$)**.** *There is a randomized quasipolynomial-time algorithm for agnostically learning, with membership queries, a function $f\colon \{0,1\}^n \to \{0,1\}$ with $\mathrm{dist}(f, \mathsf{AC}^0[2]) \leq \beta$ (for a non-negligible $\beta > 0$), producing a circuit that computes f on all but at most $\mathsf{poly}(\log n) \cdot \beta$ fraction of inputs.*

4.3.5 The case of $\mathsf{AC}^0[q]$ for prime $q > 2$

Next, we consider the case of agnostic learning for $\mathsf{AC}^0[q]$ for prime $q > 2$. While this follows the general outline of the $\mathsf{AC}^0[2]$ case, there are some differences. In particular, to keep the function generators close to functions in $\mathsf{AC}^0[q]$, we need to consider them as producing functions which take Boolean $\{1,-1\}$ inputs to outputs in the range $\{0,\ldots,q-1\}$ of integers modulo q. We need to adjust the natural property from [RR97] to handle such functions. This turns out to actually simplify the argument from [RR97] and to eliminate one step (the von Neumann construction) from the PRFG construction in [CIKK16]. Our learning algorithm follows the same general outline as above.

Preconditions We assume membership query access to a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$, and a value β and integer d so that we are promised that there is an h in $\mathsf{AC}^0[q]$ computable by a depth d circuit and $\mathrm{dist}(f,h) \leq \beta$.

Amplification Given a parameter $k = k(n)$, the mod q amplification $\mathsf{Amp}_{k,q}(f)$ is defined as the mod q Goldreich-Levin (Hadamard code) encoding of the k-wise direct product of f:

$$\textsc{Amp}_{k,q}(f) = F(x_1,\ldots,x_k,b_1,\ldots,b_k) = \sum_{i=1}^{k} b_i \cdot f(x_i),$$

where $x_1, \ldots, x_k \in \{1,-1\}^n$, $b_1, \ldots, b_k \in \{0, \ldots, q-1\}$, and the summation is modulo q.

Note that this function takes on values in $\{0, \ldots, q-1\}$. We will extend the class $AC^0[q]$ to include such functions in any of several obvious ways, e.g., by having q output gates with the one true one selecting the output. We can code inputs taking on such values similarly.

In our construction, we will set $k = 1/(10 \cdot \beta)$. Let the functions H and F be defined by $H = \mathsf{Amp}_{k,q}(h) \in \mathsf{AC}^0[q]$ and $F = \mathsf{Amp}_{k,q}(f)$. Then $\mathrm{dist}(F,H) = (1-(1-\beta)^k)(1-1/q) \leq k\beta = .01$, since if f and h agree on all k inputs, the functions F and H will agree, and otherwise, they agree with conditional probability $1/q$. Also, H is computable by a depth $d+2$ $\mathsf{AC}^0[q]$ circuit of polynomial size, and a query to F can be simulated with k queries to f.

Pseudo-random function generator As in [CIKK16], we use a version of the NW generator with a design based on polynomials over $\mathrm{GF}(q)$. We are applying this to the function F with non-Boolean outputs from $\mathrm{GF}(q)$, so the resulting truth table will be, for each seed s, a vector of values mod q. We will set the stretch L to be quasi-polynomial in n, $L = \exp(C \cdot \log^{qd+c} n)$ for some constants C and c, where we need the q in the exponent of the polylog because of the overhead of GL reconstruction for circuits with outputs in $\mathrm{GF}(2)$. Note that we can construct such a truth table with L queries to F. A subtlety is that, while we look at the sets in the design as determined by polynomials over $\mathrm{GF}(q)$, we only consider those L polynomials of degree $\ell - 1$, where $\ell = \log_2 L$, with co-efficients in $\{1,-1\}$.

Call this pseudo-random function generator using F and H respectively, and seed s, G_s^F and G_s^H. As noted in [CIKK16], for each seed s, G_s^H can be computed by $\mathsf{poly}(n)$ sized circuits of depth $d + O(1)$.

Since for a random seed s and random position I, the value F is queried at is uniform, $\mathbb{E}_s\left[\mathrm{dist}(G_s^F, G_s^H)\right] = \mathrm{dist}(F,H) \leq .01$. By Markov, we get $\Pr\left[\mathrm{dist}(G_s^F, G_s^H) \geq .1\right] \leq .1$.

Apply natural property At this point, we apply a tolerant natural property. We need a variant of natural property that applies to functions with Boolean inputs and outputs in $\mathrm{GF}(q)$. It turns out that the Razborov-Rudich [RR97] natural property for $\mathsf{AC}^0[q]$ is actually simpler in this case. We prove the following theorem after completing our application.

Lemma 41 (Tolerant natural property for $\mathsf{AC}^0[q]$)**.** *There is a* P*-natural property* $\{R_n\}_{n\geq 0}$ *with largeness* $1/2$*, and* $(.15)$*-tolerant* $\exp(\Omega(n^{1/(2d)}))$*-usefulness against* d*-depth* $\mathsf{AC}^0[q]$ *circuits computing functions* $f\colon \{1,-1\}^n \to \mathrm{GF}(q)$.

We get that at most $1/10$ of the functions G_s^F will be of high complexity, whereas a random function will be of high complexity with probability $1/2$. So testing whether a function has high complexity gives us a $\mathsf{poly}(L)$ size distinguisher with constant advantage for distinguishing G_s^F from a random function.

Converting to a predictor Using the standard hybrid argument and proof of correctness for the NW generator, we can convert this distinguisher into a predictor circuit of size $\mathsf{poly}(L)$ and advantage $\Omega(1/L)$ of predicting $F(z)$ over random guessing. (To compute this predictor, we need to query F and hence f at $\mathsf{poly}(L)$ positions; see [CIKK16]. This is the main step that requires membership queries.)

Converse of amplification Applying the converse of the generalized GL construction and the direct product theorems, we can convert this predictor circuit into one that computes f on $1-\gamma$ inputs, where $(1-\gamma)^{\Omega k} = \Omega(1/L)$. Thus, $e^{-C_1\gamma k} = C_2/L$, or $\gamma = O(\log L/k) = O(\beta \cdot \log L) = O(\beta \cdot \log^{qd+c} n)$. So we get an agnostic learner that works in time and queries quasi-polynomial in n, and with error at most $O(\log^{qd+c} n)\cdot\beta$. (Note that this assumes β is non-negligible; otherwise, the time and circuit size depend on $1/\beta$ as well).

Combining all these pieces, we have the following.

Theorem 42 (Agnostic learning of $\mathsf{AC}^0[q]$)**.** *Let* $q > 2$ *be any prime. There is a randomized quasipolynomial-time algorithm for agnostically learning, with membership queries, a function*

$f\colon \{0,1\}^n \to \{0,1\}$ *with* $\mathrm{dist}(f, \mathsf{AC}^0[q]) \leq \beta$ *(for a non-negligible* $\beta > 0$*), producing a circuit that computes* f *on all but at most* $\mathsf{poly}(\log n) \cdot \beta$ *fraction of inputs.*

4.3.6 Tolerant Natural Properties

We conclude this section by producing the required tolerant natural properties.

Tolerant natural property for $\mathsf{AC}^0[2]$

Razborov [Raz87] showed the following natural property for $\mathsf{AC}^0[2]$:

> Given an n-variate boolean function f, construct certain matrices $A_1, \ldots, A_b$, for $b = n/2 - \sqrt{n}$, of dimensions at most $2^n \times 2^n$, and check if at least one of the matrices has rank at least $2^n/(140 \cdot n^2)$ over $\mathbf{GF}(2)$.

More precisely, for $a = n/2 - \sqrt{n}$ and all $i \leq a$, define A_i to be the matrix whose rows are labeled by size a subsets of $[n]$, and whose columns are labeled by size i subsets of $[n]$. For $K \subseteq [n]$, let $Z(K) = \{x \in \{0,1\}^n \mid x|_K = \vec{0}\}$. For a row $I \subseteq [n]$ and a column $J \subseteq [n]$, define $(A_i)_{I,J} = \oplus_{x \in Z(I \cup J)} f(x)$.

It is possible to show that at least $1/2$ of all n-variate boolean functions satisfy this property; so we have largeness (see [CIKK16]). The usefulness of this property is due to the following two lemmas. Below we denote by $\mathscr{P}(D)$ the linear space of all n-variate degree D multilinear polynomials over $\mathbf{GF}(2)$.

Lemma 43 ([Raz87])**.** *For an* n*-variate boolean function* f *and the corresponding matrices* $A_1, \ldots, A_b$*, for* $b = n/2 - \sqrt{n}$*, we have for all* $1 \leq i \leq b$ *that*

$$\mathrm{DIST}(f, \mathscr{P}(\sqrt{n})) \geq \mathrm{rank}(A_i).$$

Lemma 44 ([Raz87])**.** *For an* n*-variate boolean function* f*, if* f *is computable by a* d*-depth* $\mathsf{AC}^0[2]$ *circuit of size* s*, then*

$$\mathrm{dist}(f, \mathscr{P}((O(\log(s/\varepsilon))^d)) \leq \varepsilon.$$

So for $\varepsilon = 1/n^3$ and size $s < \exp(\Omega(n^{1/(2d)}))/n^3$, we get by Lemma 44 that any f computable by a d-depth $\mathsf{AC}^0[2]$ circuit of size s is such that $\mathrm{DIST}(f, \mathscr{P}(\sqrt{n})) \leq 2^n/n^3$. Hence, by Lemma 43, all the corresponding matrices A_i for f have rank at most $2^n/n^3 \leq 2^n/(140 \cdot n^2)$ (for all sufficiently large n), and so f is rejected by the natural property.

Now suppose that h is an n-variate boolean function that is close to f, i.e., for some $0 \leq \beta \leq 1$,

$$\mathrm{dist}(h, f) \leq \beta,$$

where f is as above. Then we get by the triangle inequality that

$$\mathrm{DIST}(h, \mathscr{P}(\sqrt{n})) \leq (\beta + n^{-3}) \cdot 2^n,$$

which, in particular, means that for any $\beta \leq 1/n^3$, such a function h will also be rejected by the natural property above.

Thus we have proved the following.

Lemma 45 (Tolerant natural property for $\mathsf{AC}^0[2]$)**.** *There is a* P*-natural property* $\{R_n\}_{n\geq 0}$ *with largeness* $1/2$*, and* $(1/n^3)$*-tolerant* $\exp(\Omega(n^{1/(2d)}))$*-usefulness against* d*-depth* $\mathsf{AC}^0[2]$ *circuits.*

Tolerant natural property for $\mathsf{AC}^0[q]$ for prime $q > 2$

Here we prove the following.

Lemma 46 (Tolerant natural property for $\mathsf{AC}^0[q]$)**.** *There is a* P*-natural property* $\{R_n\}_{n\geq 0}$ *with largeness* $1/2$*, and* $(.15)$*-tolerant* $\exp(\Omega(n^{1/(2d)}))$*-usefulness against* d*-depth* $\mathsf{AC}^0[q]$ *circuits computing functions* $f\colon \{1,-1\}^n \to \mathrm{GF}(q)$.

Proof. Let $\mathscr{M}$ be the vector space of all n-variate multilinear polynomials over $GF(q)$, and let $\mathscr{L}$ be the subspace of those polynomials of degree at most $n/2$. Given such a multilinear polynomial f (and any truth table indexed by $\{1,-1\}^n$ over $GF(q)$ defines such a polynomial), we say that f is high complexity if the dimension $\dim(\mathscr{L} + f \cdot \mathscr{L}) \geq 3/4 \cdot N$, where $N = 2^n$.

Note that, for any function f of degree d, $\mathscr{L} + f \cdot \mathscr{L}$ is contained within the space of multilinear polynomials of degree $l/2+d$, which has dimension at most $N(1/2+O(d/\sqrt{n}))$. Changing any D values can increase this dimension by at most D (since adding the dimension D vector space of all functions on these D points to the subspace for the original function includes the subspace functions for the changed function). So in particular, any high complexity function must have distance at least $1/5$ from any function of degree $c\sqrt{n}$ for some $c > 0$. Since by work by Razborov [Raz87] and Smolensky [Smo87], any function in $\mathsf{AC}^0[q]$ of depth d and size s is within ε distance of a multilinear polynomial over GF(q) of degree $O(\log(s/\varepsilon)^d)$, any high complexity function must be distance .15 from any function computed by size $\exp(\Omega(n^{1/(2d+C)}))$ depth $d+C$ circuits with mod q gates.

At least half of such functions have high complexity. From [Smo87], for p the product of all l inputs (i.e., the parity of the number of -1 inputs), $\mathscr{L} + p \cdot \mathscr{L} = \mathscr{M}$. Then for f any function, either f has high complexity or $p-f$ does. Because if both have low complexity, then

$$\dim(\mathscr{L} + f \cdot \mathscr{L}) = \dim \mathscr{L} + \dim((f \cdot \mathscr{L})/\mathscr{L}) < \frac{3}{4} \cdot N,$$

so $\dim((f \cdot \mathscr{L})/\mathscr{L}) < (1/4) \cdot N$, and similarly for $p-f$. Then

$$\begin{aligned}
\dim \mathscr{M} &= \dim(\mathscr{L} + p \cdot \mathscr{L}) \\
&\leq \dim(\mathscr{L} + f \cdot \mathscr{L} + (p-f) \cdot \mathscr{L}) \\
&\leq \dim \mathscr{L} + \dim((f \cdot \mathscr{L})/\mathscr{L}) + \dim(((p-f) \cdot \mathscr{L})/\mathscr{L}) \\
&< N/2 + N/4 + N/4 \\
&= N,
\end{aligned}$$

a contradiction. Since all functions can be paired up into $f, p-f$ pairs, at least half the functions have high complexity. Clearly, we can test whether a function has high complexity in $\mathsf{poly}(N)$ time. □

4.4 Agnostic learning from tolerant natural properties

Next, we consider the case of agnostic learning for any Λ closed under $\mathsf{AC}^0[2]$-reductions using *any* natural property against Λ with super-constant tolerance and usefulness. This follows the general outline of the $\mathsf{AC}^0[2]$ case, but we need to use a variant of the NW pseudorandom generator to take advantage of the (potentially) better tolerance. We will use Chebyshev instead of Markov to bound the probability, over random seeds z, that the functions mapped to by the generator have small distance. Our generic learning algorithm also follows the outline.

Preconditions Let Λ be some complexity class closed under $\mathsf{AC}^0[2]$-reductions. Let $\mathscr{R}$ be a BPP-constructive, τ-tolerant, u-useful natural property against Λ, for super-constant τ with largeness $\delta > (1/2)$. Write $\tau = (1/2) - \tau'$, because it will sometimes be easier to work with τ as an "advantage." We assume membership query access to a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$, and a value β so that we are promised that there is an h in Λ with $\mathrm{dist}(f,h) \leq \beta$.

Amplification We use $\textsc{Amp}_k$ identically to the specific case of $\mathsf{AC}^0[2]$, except that we set k later based on abstract τ and u. Let $F = \textsc{Amp}_k(f)$, $H = \textsc{Amp}_k(h)$, as before $\mathrm{dist}(F,H) = (1/2) - (1/2)(1-\beta)^k$, which we call μ.

Pseudo-random function generator As in [CIKK16], we use a version of the NW generator with a design based on polynomials over $\mathrm{GF}(2)$. Recall that the NW design for parameters $n,m,L \in \mathbb{N}$ is a family of sets $S_1,\ldots,S_L \subseteq [m]$, of size $|S_i| = n$, for all $1 \leq i \leq L$, and small overlap $|S_i \cap S_j| \leq \log L = \ell$ for all $1 \leq i \neq j \leq L$. It was shown in [CIKK16] that such designs can be efficiently locally computed by $\mathsf{AC}^0[q]$ circuits, for any prime q.

Lemma 47 (NW design in $\mathsf{AC}^0[q]$ [NW94, CIKK16])**.** *Let q be any prime. There is a constant $d_0 \in \mathbb{N}$ such that, for any n and $L < 2^n$, there exists an NW design $S_1,\ldots,S_L$ with parameters as defined above, so that the function $MX_{NW}\colon \{0,1\}^\ell \times \{0,1\}^m \to \{0,1\}^n$,*

defined by $MX_{NW}(i,z) = z|_{S_i}$*, where* $z|_{S_i}$ *denotes the substring of* z *indexed by* S_i*, is computable by an* $\mathsf{AC}^0[q]$ *circuit of depth* d_0 *and size* $\mathsf{poly}(\ell,n)$.

The NW generator [NW94] based on a boolean function $F\colon \{0,1\}^n \to \{0,1\}$ is the new function $G^F\colon \{0,1\}^m \to \{0,1\}^L$ defined as $G^F(z) = F(z|_{S_1}) \circ \cdots \circ F(z|_{S_L})$, where $S_1,\ldots,S_L$ is the NW design as above. Lemma 47 implies that if $F \in \mathsf{AC}^0[2]$, then, for each seed z, the output $G^F(z)$ is the truth table of an $(\ell = \log L)$-variate Boolean function of $\mathsf{AC}^0[2]$ circuit complexity at most $\mathsf{poly}(\ell,n)$.

Let $H\colon \{0,1\}^n \to \{0,1\}$ be another boolean function such that $\mathrm{dist}(F,H) \le \mu$, for some $\mu \in [0,1]$. By the definition of the NW generator, we have that the expected Hamming distance between the L-bit strings $G^F(z)$ and $G^H(z)$, over random seeds z, is $\mathrm{dist}(F,H)\cdot L \le \mu \cdot L$. For our agnostic learning framework, it is important (as explained in the previous section) that the NW generator have the *concentration property*: for most seeds z, the Hamming distance between $G^F(z)$ and $G^H(z)$ is close to the expected distance $\mu \cdot L$.

We achieve this concentration property by adding a pairwise-independent string generator as a component of the NW generator. Let $PI\colon \{0,1\}^\ell \times \{0,1\}^{m'} \to \{0,1\}^n$ be a *pairwise independent generator* such that

1. for each $i \in [L]$, the distribution $PI(i,z)$ over uniformly random $z \in \{0,1\}^{m'}$ is uniform over $\{0,1\}^n$, and
2. for all $i \neq j \in [L]$, the distribution of $PI(i,z)$ and $PI(j,z)$, over uniformly random seeds $z \in \{0,1\}^{m'}$, is uniform over $\{0,1\}^n \times \{0,1\}^n$.

Such generators exist for $m' \le n(\ell+1)$; for example, pick a random $0/1$ matrix A of dimension $n \times \ell$ and a random $0/1$ vector v of dimension n. Let $z = (A,v)$. Define $P(i,(A,v)) = A \cdot i + v$, where $A \cdot i$ denotes the matrix-vector multiplication, and all operations are over $\mathbf{GF}(2)$. It is easy to see that this generator $PI(i,z)$ is computable by an $\mathsf{AC}^0[2]$ circuit of polynomial size.

Define the modified NW generator $G'^F : \{0,1\}^m \times \{0,1\}^{m'} \to \{0,1\}^L$, based on the n-variate boolean function F, as follows:

$$G'^F(z_1, z_2) = F(z_1|_{S_1} \oplus PI(1, z_2)) \circ \cdots \circ F(z_1|_{S_L} \oplus PI(L, z_2)),$$

where S_i's form the NW design, and PI is the pairwise-independent generator as above, and $\oplus$ denotes the bit-wise XOR of the corresponding n-bit strings.

Observe that since the generator PI is efficiently locally computable in $\mathsf{AC}^0[2]$, we still get (by Lemma 47) that the ℓ-bit function output by G'^F, for $F \in \mathsf{AC}^0[2]$, has $\mathsf{AC}^0[2]$ circuit complexity at most $\mathsf{poly}(\ell, n)$. Next, the generator G' allows the same kind of reconstruction as the original NW generator: given a distinguisher for G' with a constant distinguishing probability, one can efficiently construct (using membership queries to F) a small circuit computing F on at least $1/2 + \Omega(1/L)$ fraction of inputs. Finally, the generator $G'^F(z_1, z_2)$, for uniformly random seeds z_1 and z_2, outputs L values of F on *pairwise-independent* uniformly random n-bit inputs.

From pairwise independence we get that the Hamming distance between $G'^F(z_1, z_2)$ and $G'^H(z_1, z_2)$, over random z_1 and z_2, is concentrated around the expectation, by the Chebyshev bound. More precisely, for F and H with $\mathrm{dist}(F, H) \leq \mu$, we have by Chebyshev that

$$\mathbf{Pr}_z \left[\left| \mathrm{DIST}(G'^F(z), G'^H(z)) - \mu \cdot L \right| > \zeta \cdot L \right] < \frac{1}{\zeta^2 \cdot L},$$

which we will require to be less than $1/4$. We parameterize the bound with $\zeta = (1/4)(1 - \beta)^k$. For the selected ζ, and the stretch L we are forced to pick later, this is immediate.

Apply natural property At this point, we apply a tolerant natural property to produce a distinguisher circuit for the generator above. This induces the following system of constraints, which relate the usefulness, tolerance, and density of the property to the stretch and concentration of the generator. Let $\Lambda\text{-SIZE}(G'^H(z)) = s_H$. We require that $s_H \leq u(\ell)$, to

respect the size lower bound. We re-arrange the Chebyshev bound above and see that we should require $\mu + \zeta < \tau(\ell)$ to respect tolerance and ensure a good distinguishing gap from the property. We satisfy the first requirement by setting $\ell \geq u^{-1}(s_H)$. The second one is equivalent to $(1/4)(1-\beta)^k > \tau'(\ell)$. In this case, the tolerant property can only accept $G^F(z)$ with probability (1/4) but accepts a random function with probability at least (1/2), giving us a (1/4) distinguishing gap. We can satisfy both constraints by setting $k = \Theta(\log(\tau'(\ell))/\beta)$.

Converting to a predictor Using a small modification of the standard hybrid argument and proof of correctness for the NW generator, we can convert this distinguisher into a predictor circuit of size $\mathsf{poly}(L)$ and advantage $\Omega(1/L)$ of predicting $F(z)$ over random guessing. The modified predictor just embeds a construction of PI and shifts/unshifts inputs to the distinguisher circuit as necessary. (To compute this predictor, we need to query F and hence f at $\mathsf{poly}(L)$ positions; see [CIKK16]. This is the main step that requires membership queries.) From this step we know that our runtime is at most $\mathsf{poly}(L)$, and the circuit output at this stage is already size $\mathsf{poly}(L)$.

Converse of amplification Identical to the case of $\mathsf{AC}^0[q]$, but with the additional constraints mentioned above. Note that the runtime of these algorithms is randomized time in the size of the input circuit, so runtime, number of queries, and output circuit size of this stage will also be dominated by L. Use of this algorithm imposes the following constraint from the direct product reconstruction stage: $\mathsf{poly}(1/L) > e^{-k\varepsilon/c}$. So $\varepsilon > \Theta(\log(L)/k)$. Substituting in our value for k, this gives us $\varepsilon = \Theta(\ell\beta/\log(\tau'(\ell)))$ for $\ell = u^{-1}(s_H)$.

Summarizing, we get a generic reduction from tolerant natural properties to agnostic learning.

Theorem 48 (Tolerant natural properties imply agnostic learning algorithms)**.** *Let $\mathscr{R}$ be a natural property against Λ closed under $\mathsf{AC}^0[2]$ reductions with $(1/2-\tau')$-tolerant u-usefulness and*

largeness $\delta \geq 1/2$. *Then there is a randomized algorithm such that, for any n-ary boolean functions* f *and* h *with* $\mathrm{dist}(f,h) < \beta$ *and* $s_h = \Lambda$-$SIZE(h)$, *the algorithm, given oracle access to* f, *produces a circuit* ε-*approximating* f, *for any* $\varepsilon > \beta \cdot u^{-1}(\mathsf{poly}(s_h))/\log(\tau'(u^{-1}(\mathsf{poly}(s_h))))$, *in time* $\mathsf{poly}(\max\{2^{u^{-1}(\mathsf{poly}(s_h))}, 1/\varepsilon\})$.

In particular, this means if we have a "perfect" natural property, with exponential usefulness u and inverse exponential tolerance τ', we have a polynomial-time learning algorithm with error bound $\Theta(\beta)$. Thus Theorem 35 is a special case of Theorem 48.

4.5 Hardness of removing membership queries

Is it possible to eliminate membership queries from our algorithm, learning just from random examples? We note that removing membership queries would give us quasipolynomial-time algorithms for two notoriously difficult problems: learning parities with noise (LPN) for the case of $\mathsf{AC}^0[2]$ and a variant of learning with errors (LWE) for $\mathsf{AC}^0[q]$.

Though learning parities with noise under uniform distribution can be done in polynomial time with membership queries (by the Goldreich-Levin algorithm [GL89]), without membership queries this problem is believed to be hard. Learning parities with noise efficiently under uniform distribution would give learning algorithms for DNFs and k-juntas (and in general, for any problem reducible to finding a heavy Fourier coefficient of a function) [FGKP06].

In the worst case, LPN is known to be NP-hard (and MAX-SNP-hard). The average-case hardness of LPN has been considered as early as 1993, when Blum, Furst, Kearns and Lipton have given a simple construction of a pseudorandom bit generator based on the assumption that learning parities with constant noise rate is hard [BFKL93]. In practical cryptography, average-case hardness of LPN is the basis for Hopper and Blum authentication protocol [HB01]. There, the noise rate is usually set to a constant $\eta \in (0, 1/2)$, in particular $\eta = 1/8$ has been used in applications [LF06]. Though for $\mathsf{AC}^0[2]$ our algorithm works for noise up to $1/\mathsf{polylog}(n)$, we can tolerate constant noise for $\mathsf{AC}^0[q]$.

Hardness of LWE problem follows from worst-case hardness of variants of the lattice shortest vector problem [Reg09]. Whereas LPN has been used to build "minicrypt" cryptographic primitives, LWE has been used for public-key cryptosystems [Ale03, Reg09].

4.6 Open questions

While there are correlation bounds for $\mathsf{AC}^0[q]$ circuits that say that some explicit functions cannot be computed by "small" circuits on significantly more that $1/2 + 1/\sqrt{n}$ fraction of inputs, we do not know how to get natural properties with tolerance close to $1/2$. Getting natural properties with better tolerance parameters would immediately imply improved parameters for our agnostic learning algorithms for the corresponding circuit classes. (Of course, getting stronger correlation bounds for $\mathsf{AC}^0[q]$, whether obtained by natural proofs or not, is in itself a very important problem in circuit complexity.)

Can one get a query agnostic learning algorithm for $\mathsf{AC}^0[q]$ with the optimal error $\mathsf{opt} + \varepsilon$? It seems that, even with ideal tolerance and usefulness, our approach of getting learning algorithms from natural properties will at best achieve the error $O(\mathsf{opt}) + \varepsilon$. So one needs a new approach, perhaps inspired by the learning algorithm in this chapter.

In fact, probably the main open problem is to get a more "natural" (understandable) learning algorithm for $\mathsf{AC}^0[q]$ than our construction, which combines the NW-style generator analysis with circuit lower bound proofs. As a possible first step, it would be interesting to get an alternative agnostic learning algorithm for low-degree polynomials over $\mathrm{GF}(2)$.

Chapter 4, in part, is based on the material as it appears in "Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Agnostic learning from tolerant natural proofs. In Klaus Jansen, José D. P. Rolim, David Williamson, and Santosh Srinivas Vempala, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, volume 81 of *LIPIcs*, pages 35:1–35:19. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017". The

dissertation author was the primary investigator and author of this paper.

Chapter 5

FG Derandomization

In this chapter, we show that popular hardness conjectures about problems from the field of fine-grained complexity theory imply structural results for resource-based complexity classes. Namely, we show that if either k-Orthogonal Vectors or k-CLIQUE requires $n^{\varepsilon k}$ time, for some constant $\varepsilon > 1/2$, to *count* (note that these conjectures are significantly weaker than the usual ones made about those problems) on randomized machines for all but finitely many input lengths, then we have the following derandomizations:

- BPP can be decided in polynomial time *using only n^{α} random bits* on average over *any* efficient input distribution, for any constant $\alpha > 0$
- BPP can be decided in polynomial time with *no randomness* on average *over the uniform distribution*

This answers an open question of Ball et al. (STOC '17) in the positive of whether derandomization can be achieved from conjectures about fine-grained complexity theory. More strongly, these derandomizations improve over all previous results achieved from *worst-case uniform* assumptions by succeeding on all but finitely many input lengths. Previously, derandomizations from worst-case uniform assumptions were only know to succeed on infinitely many input lengths. It is specifically the structure and moderate hardness of the k-Orthogonal Vectors and k-CLIQUE problems that makes removing this restriction possible.

Via this uniform derandomization, we connect the problem-centric and resource-centric views of complexity theory by showing that exact hardness assumptions about specific problems like k-CLIQUE imply quantitative and qualitative relationships between randomized and deterministic time. This can be either viewed as a barrier to proving some of the main conjectures of fine-grained complexity theory lest we achieve a major breakthrough in unconditional derandomization or, optimistically, as route to attain such derandomizations by working on very concrete and weak conjectures about specific problems.

5.1 Introduction

Computational complexity can be viewed through two main perspectives: problem-centric or resource-centric. Problem-centric complexity theory asks what resources are required to solve *specific problems*, while resource-centric complexity deals with the relative power of different computational models given different resource budgets such as time, memory, non-determinism, randomness, circuit size, etc. (see [GI16] for a discussion). Through complete problems, these two perspectives often coincide, so that a resource-centric view acts as a fine proxy for answering questions about the complexity of specific problems. The rapidly progressing field of fine-grained complexity theory, however, brings attention back to the problem-centric viewpoint, raising fine distinctions even between problems complete for the same complexity class, and making connections between problems at very different levels of complexity. To what extent are these two approaches linked, i.e., to what extent can inferences about the fine-grained complexities of specific problems be made from general assumptions about complexity classes, and vice versa?

Here, we examine such links between the fine-grained complexity of specific problems such as the k-Orthogonal Vectors and k-CLIQUE problems and general results about derandomization of algorithms. Derandomization has been a very fruitful study in complexity theory, with many fascinating connections between lower bounds, showing that problems require large amounts of resources to solve, and upper bounds, showing that classes of probabilistic algorithms

can be 'derandomized' by simulating them deterministically in a non-trivial fashion. In particular, the hardness-to-randomness framework shows that in many cases, the existence of *any* "hard" problem can be used to derandomize classes of algorithms. We reconsider this framework from the fine-grained, problem-centric perspective. We show that replacing a generic hard problem with *specific* hardness conjectures from fine-grained complexity leads to quantitatively and qualitatively stronger derandomization results than one gets from the analogous assumption about a generic problem. In particular, we show that starting from these assumptions, we can simulate any polynomial-time probabilistic algorithm (on any samplable distribution on inputs with a very small fraction of errors) by a polynomial time probabilistic algorithm that uses only n^{α} random coins, for any $\alpha > 0$. This type of derandomization previously either assumed the existence of cryptographic One-Way Functions or *exponential non-uniform* hardness of Boolean functions; our assumptions are both much weaker and quite popular.

Thus, the problem-centric conjectures of fine-grained complexity cannot live in isolation from classical resource-centric consequences about the power of randomness. Viewed another way, our results can be seen as a barrier to proving some of the key hardness assumptions used by fine-grained complexity theory. That is, despite recent progress towards proving hardness for k-Orthogonal Vectors, one of fine-grained complexity's key problems, in restricted models of computation [KW17], doing so for general randomized algorithms would immediately prove *all* problems in BPP are easy on average (over, say, uniformly chosen inputs). Alternatively, the reader may choose to be optimistic. "All" we need to do for efficient derandomization is prove uniform lower bounds for a highly-structured family of problems that live inside P.

Previous derandomization results in the uniform setting ([IW01, GW02, TV07]) used two properties of the hard problem: random self-reducibility and downward self-reducibility. To obtain our results, we need problems that have stronger, "fine-grained" versions of both properties (or can be reduced to problems that do). In particular, we need problems where not only can instances of size n be reduced to smaller instances of the same problem, but that these instances are much smaller, of size n^{ε} for $\varepsilon < 1$, and that the reduction is "fine-grained", in that

any improvement in the time to solve the smaller instances yields a similar improvement in the time to solve the larger ones. Luckily, the "core" problems studied in fine-grained complexity have exactly the desired properties.

5.1.1 Our Results

We obtain two main theorems about the power of BPP from uniform worst-case assumptions about well-studied problems from fine-grained complexity theory. We consider the k-Orthogonal Vectors (k-OV) and the k-CLIQUE problems, defined and motivated in Section 5.2.1, and show that (even weaker versions of) popular conjectures about their hardness give two flavors of average-case derandomization that improve over the classical *uniform* derandomizations.

All previous derandomizations from *uniform* assumptions on worst-case hardness only succeed on *infinitely many input lengths*. Our work is the first to use worst-case uniform assumptions to derandomize BPP *for all but finitely many input lengths*, giving a standard inclusion. The only other worst-case uniform assumptions known to imply such results are those so strong as to imply cryptographic assumptions or circuit lower bounds, hewing closer to the cryptographic or non-uniform derandomization literature. In contrast, our *uniform derandomizations* are from extremely weak worst-case uniform conjectures on simple, natural, combinatorial problems. Informally, we prove the following:

Informal Theorem 1 (see Theorem 75 on page 124)**.** *If k-OV or k-CLIQUE requires $n^{\varepsilon k}$ time, for some constant $\varepsilon > 1/2$, to **count** on randomized machines in the worst-case for all but finitely many input lengths, then* BPP *can be decided in polynomial time **using only** n^{α} **random bits** on average over **any** efficient input distribution, for any constant $\alpha > 0$.*

Randomness can be removed entirely by simply brute-forcing all random bits and taking the majority of the outputs to give the following more standard full derandomization.

Corollary. *If k-OV or k-CLIQUE requires $n^{\varepsilon k}$ time, for some constant $\varepsilon > 1/2$, to **count** on randomized machines in the worst-case for all but finitely many input lengths, then* BPP *can*

*be decided with **no randomness** in sub-exponential time on average over **any** efficient input distribution.*

This conclusion is strictly stronger than the classic uniform derandomizations of [IW01, TV07]. The weakest uniform assumption previously known to imply such a conclusion was from those already strong enough to imply the cryptographic assumption of the existence of One-Way Functions that are hard to invert for polynomial time adversaries [BM84, GKL93, GL89, HILL99, Yao82] or those implying *non-uniform* circuit lower bounds [BFNW93].

Our second main theorem, using techniques from [KvMS12], shows how to remove all randomness *within polynomial time* if the distribution over inputs is uniform. The only stronger derandomizations from uniform assumptions were, again, from assumptions already strong enough to imply circuit lower bounds or from the cryptographic assumption of the existence of One-Way Permutations that require *exponential* time to invert [BM84, GL89, Yao82].

Informal Theorem 2 (see Theorem 80 on page 127)**.** *If k-OV or k-CLIQUE require $n^{\varepsilon k}$ time, for some constant $\varepsilon > 1/2$, to **count** on randomized machines in the worst-case for all but finitely many input lengths, then* BPP *can be decided in polynomial time with **no randomness** on average over **the uniform distribution**.*

These results should be viewed through three main points: First, that we conceptually tie problem-centric conjectures to resource-centric consequences, thus partially reconnecting the two perspectives on complexity theory that separate in the fine-grained world. Secondly, we add to the general derandomization literature by achieving quantitatively and qualitatively stronger derandomization from weak uniform assumptions. Lastly, our results can be seen, pessimistically, as demonstrating a barrier to proving even weak versions of some of fine-grained complexity theory's main conjectures lest we achieve a breakthrough in unconditional derandomization or, optimistically, as providing a path to achieve such general resource-centric results by instead considering extremely weak conjectures on very concrete, simple, and structured combinatorial problems. The reader may determine their own level of optimism.

5.1.2 Related Work

We now discuss previous connections between problem-centric and resource-centric complexity and previous derandomization results.

Connections Between Problem-Centric and Resource-Centric Complexity.

Most connections from problem-centric to resource-centric complexity show that faster algorithms for OV or related problems give circuit lower bounds. For instance, improvements in EDIT-DISTANCE algorithms imply circuit lower bounds [AHWW15] and solving OV faster (and thus CNF-SAT [Wil05]) implies circuit lower bounds [JMV15]. These are all non-uniform results, however, whereas in this paper we are concerned with *machines* and their resource-bounds as opposed to circuits. On the uniform side, [GIKW17] recently showed that the exact complexity of k-Orthogonal Vectors is closely related to the complexity of uniform AC^0, although a connection between more powerful machine models and fine-grained assumptions was still not known until now. Further, all of these connections follow from OV being *easy*. Our work shows that there are also interesting resource-centric consequences if our fine-grained problems are *hard*.

Uniform Derandomization Framework.

The uniform derandomization framework was introduced in [IW01], a breakthrough paper that showed the first derandomization from a uniform assumption ($\mathsf{EXP} \neq \mathsf{BPP}$) in the *low-end* setting: a weak assumption gives a *slow* (but non-trivial, ie, subexponential-time) deterministic simulation of BPP. This is in contrast to our simulation which retains small amounts of randomness but is *fast* (this is a strictly stronger result as it recovers the [IW01] derandomization as a corollary).

The techniques introduced by [IW01] (and used in all subsequent works, including ours) for uniform derandomization seem to inherently result in simulations that are sometimes incorrect, but successfully *hide these errors from any efficient adversary*. That is, the uniform derandomizations are, for infinitely many input lengths, "pseudo-time simulations" (see Defini-

tion 61), meaning that the deterministic simulation may make some errors, but such bad inputs are intractable to sample. If a simulation fails in the woods, but nobody is smart enough to hear it, then is it really wrong?

We build on [TV07], which simplifies the proof of [IW01] to prove that PSPACE $\neq$ BPP implies a non-trivial deterministic simulation of BPP. The technique of [TV07] carefully arithmetizes the PSPACE-complete problem TQBF and uses this as a hard function in the generator of [IW01]. Our proof substitutes a carefully-arithmetized k-OV problem from [BRSV17]. Numerous other works study derandomization from uniform assumptions ([Kab01, Lu01, IKW02, GST03, SU09]), but these all focus on assumptions and consequences about *nondeterministic* classes.

All worst-case uniform derandomizations, including [TV07] and [IW01], seem to only be able to achieve simulations of BPP that succeed for infinitely many input lengths because of how their proofs use downward self-reductions. Our is the first work to achieve simulations on *all but finitely many input lengths*, because the k-OV and k-CLIQUE-inspired problems have very parallelizable downward self reductions; we can reduce to a *single* much smaller input length rather than recurse through a chain of incrementally smaller input lengths in the downward self-reduction.

Heuristics by Extracting Randomness From the Input.

A separate line of work began when [GW02] introduced the idea of using the *input itself* as a source of randomness to heuristically simulate randomized algorithms over uniformly-distributed inputs. If life gives you random bits, you may as well make PRGs. While their assumptions contain oracles and are mostly non-uniform and average-case, they construct an algebraic problem inside P whose worst-case uniform hardness can be used in the framework of [IW01] to get an *infinitely-often* simulation of BPP in polynomial time. Our work differs in that: (i) we achieve an *almost-everywhere* simulation, (ii) our assumptions are based on canonical fine-grained problems, and (iii) our assumptions aren't against machines with SAT-oracles.

Further, the downward self-reduction of their problem requires an expansion by minors of the determinant and so they cannot also obtain an *almost-everywhere* heuristic using our techniques without placing the determinant in NC^1 (as our modification to [IW01] exploits *embarrassingly parallel* downward self reductions). It appears *very* unlikely that the determinant is in NC^1.

The work of [KvMS12], generalizing [Sha11], removes the SAT-oracles needed in the assumptions of [GW02] by showing that the Nisan-Wigderson generator (see [NW94]) remains secure against *non-uniform* adversaries even if the seed is revealed to potential distinguishers. In Section 5.4.2 we will show their arguments can be made *uniform* so we can derandomize from uniform assumptions. Seed-revealing Nisan-Wigderson generators are used in [KvMS12] to obtain polynomial-time heuristics for randomized algorithms, where the uniformly distributed input is used as a seed to the generator. The derandomizations in [KvMS12] are achieved from *non-uniform* assumptions of polynomial *average-case* hardness. From *worst-case* and *uniform* assumptions we achieve the same derandomizations.

Organization

- §5.2: We introduce conjectures from fine-grained complexity, basic notions of average-case complexity, and the seed-extending generators of [KvMS12].
- §5.4: We give a strategy for computing k-OV, given that distinguishers for a seed-extending PRG based on k-OV are uniform. Then, we show how to uniformly generate distinguishers for seed-extending PRGs if they fail to derandomize BPP.
- §5.5: We apply the derandomization to show $\mathsf{BPP} \neq \mathsf{EXP}$ under popular conjectures.
- §5.6: We conclude with some open problems about structural complexity inside P

5.2 Preliminaries

Here we give the relevant background from fine-grained complexity theory to motivate our assumptions, present standard tools from derandomization, and explain the peculiarities that

arise specifically in derandomization from uniform assumptions such as average-case tractability and infinitely-often qualifiers.

All complexity measures of fine-grained problems will refer to time on a randomized word RAM with $O(\log(n))$-bit word length, as is standard for the fine-grained literature [Wil15, BRSV17]. Specifically, we will consider such time classs with two-sided bounded error as in [BRSV17].

We use a convention from [TV07], calling a function $t : \mathbb{N} \to \mathbb{R}^+$ a *nice time bound* if $n \leq t(n) \leq 2^n$, $t(n)$ is non-decreasing, $t(n)$ is computable in time $\mathsf{poly}(n)$, and $t(O(n)) \leq \mathsf{poly}(t(n)) \leq t(\mathsf{poly}(n))$. All functions of the form n^c, $n^c \log(n)$, $2^{\log^c(n)}$ and 2^{cn} satisfy these conditions.

5.2.1 Fine-Grained Hardness Conjectures

The problem-centric field of fine-grained complexity theory has had impressive success in showing the fixed polynomial time ("fine-grained") hardness of many practical problems by assuming the fine-grained hardness of four "key" well-studied problems, as discussed in [BRSV17]. We obtain our results under hardness conjectures about two of these four key problems: the k-Orthogonal Vectors (k-OV) problem and the k-CLIQUE problem. Evidence continues to accumulate that these problems are actually hard. Thus, not only is the connection between problem-centric and resource-centric complexity of independent interest, but the strong derandomizations of this paper are now supported by plausible conjectures about concrete problems.

k-CLIQUE.

Denote the matrix multiplication constant by ω. The fastest known algorithm for deciding if a graph has a k-CLIQUE (given its adjacency matrix) runs in time $O(n^{\omega k/3})$, and was discovered in 1985 [NP85] for k a multiple of three (for other k different ideas are needed [EG04]). It is conjectured that no algorithm can improve this exponent to a better constant. The parameterized

version of the famous NP-hard MAX-CLIQUE problem [Kar72], k-CLIQUE is one of the most heavily studied problems in theoretical computer science and is the canonical intractable (W[1]-complete) problem in parameterized complexity; see [ABW15] for a review of the copious evidence of k-CLIQUE's hardness and consequences of its algorithm's exponent being improved. Recent work has shown that conjecturing k-CLIQUE to require $n^{\omega k/3-o(1)}$ time, for k a multiple of three, leads to interesting hardness results for other important problems such as parsing languages and RNA folding [ABW15, BGL17, BDT16, BT17], and it is known that refuting this conjecture deterministically would give a faster exact algorithm for MAX-CUT [Wil05]. Our results hold under a *much weaker version of the conjecture*:

Definition 49 (Weak k-CLIQUE Conjecture)**.** There exists an absolute constant $\varepsilon_0 > 1/2$ such that, for all $k \in \mathbb{N}$ a multiple of three, any randomized algorithm that *counts* the number of k-CLIQUE's in an n node graph requires $n^{\varepsilon_0 k}$ time.

Note that this conjecture gives leeway for the exponent of the k-CLIQUE algorithm to be improved so long as it doesn't get down to $k/2$; even finding a linear time algorithm for Boolean matrix multipliaction ($\omega = 2$) would not contradict our conjecture! Further, even if it is possible to *decide* the k-CLIQUE problem that quickly, this conjecture still holds unless it is possible to *count all of the* k-CLIQUE*'s in that time*. With a more careful analysis of our techniques to focus on k-CLIQUE we can actually use the even weaker conjecture that $\varepsilon_0 > \omega/(\omega+3)$, as argued in Section 5.3.

k-Orthogonal Vectors.

Although the k-CLIQUE problem is certainly *at least* as important as the k-OV problem, for concreteness we will use the k-OV problem to demonstrate our techniques throughout the paper. Proofs based on hardness of k-CLIQUE follow identically.

Definition 50 (k-Orthogonal Vectors Problem, k-OV$_{n,d}$)**.** For an integer $k \geq 2$, the k-OV$_{n,d}$ problem on vectors of dimension d is to determine, given k sets $(U_1, \ldots, U_k)$ of n vectors from

$\{0,1\}^d$ each, whether there exist $u_i \in U_i$ for each i such that over $\mathbb{Z}$,

$$\sum_{\ell \in [d]} u_{1\ell} \cdots u_{k\ell} = 0$$

If left unspecified, d is taken to be $d(n) = \lceil \log^2 n \rceil$.

Definition 51 (k-Orthogonal Vectors Conjecture, k-OVC)**.** For any $d = \omega(\log n)$, for all $k \geq 2$, any randomized algorithm for the k-$\mathsf{OV}_{n,d}$ problem requires $n^{k-o(1)}$ time.

For $k = 2$ the Orthogonal Vectors conjecture for deterministic algorithms has been extensively studied and is supported by the *Strong Exponential Time Hypothesis* (SETH) [Wil05], which states that there is no $\varepsilon > 0$ such that t-SAT can be solved in time $\widetilde{O}(2^{n(1-\varepsilon)})$ for all values of t. The natural generalization to k-OV is studied in [BRSV17, GIKW17] and its deterministic hardness is also implied by SETH — therefore the deterministic k-OV conjecture is weaker than SETH.

The k-Orthogonal Vectors Conjecture is plausible independent of beliefs about the complexity of SAT: k-OVC holds unless *all* first-order graph properties become easy to decide [GIKW17] and the 2-OV conjecture has recently been proven unconditionally when the model of computation is restricted to branching programs [KW17]. The k-OVC has also been used to explain the hardness of many practical and well-studied fine-grained problems such as Local String Alignment, String Edit Distance, Dynamic Time Warping, and many other rich string problems [AWW14, BI15, BK15]. Therefore, our inability to produce truly subquadratic algorithms for all these problems despite extensive study supports the k-OVC. Furthermore, as with k-CLIQUE, our main results will hold *using a much weaker version of the randomized* k-OV *conjecture* introduced below.

Definition 52 (Weak k-Orthogonal Vectors Conjecture)**.** For any $d = \omega(\log n)$, there exists an absolute constant $\varepsilon_0 > 1/2$ such that, for all $k \geq 2$, any randomized algorithm *counting* the number of k-$\mathsf{OV}_{n,d}$ solutions requires $n^{\varepsilon_0 k}$ time.

Remark 53. *For all of these conjectures we will also consider the strengthened versions that assume that all algorithms running in time less than what is required will fail* ***on all but finitely many input lengths****, as opposed to only on infinitely many input lengths. For most natural problems, an 'almost-everywhere' assumption like this seems natural. That is, we don't expect that the problem becomes easy for, say, even input sizes and hard on odd input sizes or other degenerate cases like this, but instead believe that the hardness comes from the structure of the problem and will simply grow (as opposed to oscillate) asymptotically.*

5.2.2 Derandomization

We now define pseudorandom generators (PRGs) in terms of their distinguishers. We recall a concrete notion of distinguishing some transducer from true randomness from Chapter 2 below.

Definition 54 (Distinguishers)**.** A test $T : \{0,1\}^{m^\ell} \to \{0,1\}$ is an ε*-distinguisher* against $G : \{0,1\}^m \to \{0,1\}^{m^\ell}$, denoted $T \in \mathsf{DIS}(G,\varepsilon)$, if:

$$\left| \Pr_{r \sim \mathscr{U}_{m^\ell}} [T(r)] - \Pr_{z \sim \mathscr{U}_m} [T(G(z))] \right| > \varepsilon$$

We also will consider the seemingly weaker object of distinguishers that succeed if they are also given the seed to the PRG. These were studied in [TV07] to relate uniform derandomization to average-case hardness and in [KvMS12] to derandomize over the uniform distribution by *using the random input itself as the seed* to the PRG.

Definition 55 (Seed-Aware Distinguishers)**.** A test $T : \{0,1\}^m \times \{0,1\}^{m^\ell} \to \{0,1\}$ is an ε*-seed-aware distinguisher* against G, denoted $T \in \mathsf{DIS}(G,\varepsilon)$, if:

$$\left| \Pr_{x \sim \mathscr{U}_m, r \sim \mathscr{U}_{m^\ell}} [T(x,r)] - \Pr_{x \sim \mathscr{U}_m} [T(x,G(x))] \right| > \varepsilon$$

Standard hardness-to-randomness arguments based on presumabely "hard" function

typically prove the contrapositive: if derandomization fails, then a distinguisher for the generator can be produced. Further, from a distinguisher, we can create a small circuit for the supposedly hard function that the generator was based on. For our purposes, we require an algorithmic version of this argument for derandomization from *uniform* hardness assumptions. More specifically, we will use the following lemma[1] originally proved for distinguishers [TV07, IW01] then extended to seed-aware distinguishers by Lemma 2.9 of [KvMS12]. While the proof of [KvMS12] is non-uniform, it is easy to see that it can be made constructive, in the same way that [IW01] gave a constructive version of [NW94]. Thus, $\mathsf{DIS}(G,\varepsilon)$ in the lemma below can refer to either regular or seed-aware distinguishers (which justifies overloading this notation).

Lemma 56 (Algorithmic Distinguishers to Predictors ([TV07, IW01]))**.** *For every random self-reducible f, there exists a function G with stretch m bits to m^ℓ bits and a constant c such that*

- *$G(z)$ can be computed in time $(|z|^\ell)^c$, given oracle access to f on inputs of length at most $|z|$*
- *There exists a polynomial-time randomized algorithm A that, with high probability, given as input circuit $D \in \mathsf{DIS}(G,\varepsilon)$ for ε at least inverse polynomial and an oracle for f, prints a circuit computing f exactly.*

5.2.3 Uniform Derandomization

Previous techniques for derandomizations from worst-case uniform assumptions seemed to have inherent caveats: the derandomization only succeeds on average and, even then, only for infinitely many input lengths. Our results *will remove the infinitely-often caveat* and so, in this section, we pay careful attention to infinitely-often simulation. First, we give the definitions of average-case tractability that arise naturally from uniform derandomization.

[1] A version of which was discussed in detail by Chapter 2

Average-Case Tractability.

We begin with standard definitions of average-case tractability. For an extensive survey of these notions, see [BT06a].

Definition 57 ($t(n)$-Samplable Ensemble)**.** An ensemble $\mu = \{\mu_n\}$ is $t(n)$-samplable if there is a randomized algorithm A that, on input a number n, outputs a string in $\{0,1\}^*$ and:

- A runs in time at most $t(n)$ on input n, regardless of its internal coin tosses
- for every n and for every $x \in \{0,1\}^*$, $\Pr[A(n) = x] = \mu_n(x)$

With this notion of a samplable ensemble we can now group distributions according to their complexity. We denote by $\mathsf{SAMP}[t(n)]$ the set of all $t(n)$-samplable ensembles. Next, we can consider *heuristic algorithms* that perform "well" on a language $\mathscr{L} : \{0,1\}^* \to \{0,1\}$ when inputs are sampled according to the distribution μ. The pair $(\mathscr{L}, \mu)$ is called a *distributional problem.* We now require a notion of "tractability" for distributional problems.

As in the worst-case setting, polynomial resource bounds will be considered efficient. So, intuitively, we define HeurP as the class of distributional problems that can be solved in deterministic polynomial with negligible probability of error. Thus, while $(\mathscr{L}, \mu) \in$ HeurP is not a *worst-case* guarantee about the easiness of $\mathscr{L}$, HeurP still captures a very strong real-world notion of tractability: $\mathscr{L}$ can be easily decided up to *any* inverse polynomial probability of error when inputs are sampled from μ. It is then interesting to see over input distributions over which a language is tractable. Formally:

Definition 58 (Heuristics for Distributional Problems)**.** For time-bound $t : \mathbb{N} \to \mathbb{N}$ and error-bound $\delta : \mathbb{N} \to \mathbb{R}^+$, we say $(\mathscr{L}, \mu) \in \mathrm{Heur}_{\delta(n)}\mathsf{DTIME}[t(n)]$ if there is a time $t(n)$ deterministic algorithm A such that, for all but finitely many n:

$$\Pr_{x \sim \mu_n}[A(x) \neq \mathscr{L}(x)] \leq \delta(n)$$

For a class of languages $\mathscr{C}$ we say $(\mathscr{C},\mu) \subseteq \mathrm{Heur}_{\delta(n)}\mathsf{DTIME}[t(n)]$ if the inclusion $(\mathscr{L},\mu) \in \mathrm{Heur}_{\delta(n)}\mathsf{DTIME}[t(n)]$ holds for all $\mathscr{L} \in \mathscr{C}$.

As in [BT06a], define $\mathrm{Heur}_\delta\mathsf{P}$ as the union over all polynomials p of $\mathrm{Heur}_\delta\mathsf{DTIME}[p(n)]$ and HeurP as the intersection over all inverse polynomial $\delta(n)$ of $\mathrm{Heur}_\delta\mathsf{P}$. HeurSUBEXP is defined similarly where $\mathsf{SUBEXP} = \cap_{\varepsilon>0}\mathsf{DTIME}\left[2^{n^\varepsilon}\right]$.

To describe the *randomness-reduced* simulations we construct, we define BPTIME with a limited number of random coins in the natural way.

Definition 59 (Randomized Time with Bounded Coins)**.** We say that a language $\mathscr{L}$ is in Randomized Time $t(n)$ with $r(n)$-Bounded Coins, denoted $\mathscr{L} \in \mathsf{BPTIME}_{[r(n)]}[t(n)]$, if there is a randomized algorithm A running in time $t(n)$ and flipping $r(n)$ coins such that:

$$\forall x \in \{0,1\}^n \Pr_{r\sim\mathscr{U}_{r(n)}}[A(x;r) \neq L(x)] \leq 1/3$$

Coin-bounded random time can be combined with the notion of a heuristic in the natural way:

Definition 60 (Randomized Heuristics with Bounded Coins)**.** For time-bound $t : \mathbb{N} \to \mathbb{N}$, error-bound $\delta : \mathbb{N} \to \mathbb{R}^+$, and coin-bound $r : \mathbb{N} \to \mathbb{N}$ we say $(\mathscr{L},\mu) \in \mathrm{Heur}_{\delta(n)}\mathsf{BPTIME}_{[r(n)]}[t(n)]$ if there is randomized algorithm A running in time $t(n)$ and flipping $r(n)$ coins such that, for all but finitely many n:

$$\Pr_{x\sim\mu_n}\left[\Pr_{r\sim\mathscr{U}_{r(n)}}[A(x,r) \neq \mathscr{L}(x)] > 1/3\right] \leq \delta(n)$$

For example, $\mathrm{HeurBPP}_{[r(n)]}$ denotes the class of distributional problems that, for every inverse polynomial error, have a polynomial time randomized algorithm using only $r(n)$ random coins.

Heuristics are allowed to be distribution-specific. That is, for each distribution of interest, we may construct a different algorithm depending on the distribution. This ordering of quantifiers makes heuristics a *weak* notion of average-case tractability. A *strong* notion of average-case

tractability would require a single algorithm to succeed on every "reasonable" distribution. This intuition is captured by the standard pseudo-algorithm notion, defined formally below.

Definition 61 (Deterministic Pseudo-Time)**.** Let f be a Boolean function and let $t(n)$ be a nice time bound. We say $f \in \text{PseudoDTIME}[t(n)]$ if there is a time $t(n)$ deterministic algorithm A such that for every a and b, for all $\mu \in \text{SAMP}[n^b]$ and sufficiently large n,

$$\Pr_{x \sim \mu_n}[A(x) \neq f(x)] \leq 1/n^a$$

As a and b are arbitrary in the definition of PesudoTIME, this gives a strong notion of average-case easiness since the error must be smaller than *any* inverse-polynomial fraction for *any* polynomially samplable distribution. Thus, while errors may occur in this notion of tractability, we cannot efficiently sample them.

Infinitely-Often Simulation.

As opposed to an algorithm that decides a language (possibly on average) "for all but finitely many n" as in Definition 58, an infinitely-often (io-) qualifier can be added to any complexity class to specify that an algorithm need only succeed on infinitely many input lengths within the time and error bounds. Thus, to derandomize BPP into io-HeurP over the uniform distribution is to say that every language in BPP can be simulated on average in polynomial time by an algorithm that is only guaranteed to succeed for infinitely many input lengths. *There is no guarantee on what those input lengths are or how large the gaps could be between them.* This is obviously a very weak notion of "tractability."

Non-uniform hardness to randomness trade-offs can derandomize almost-everywhere (the desired notion of tractability for asymptotic analysis) by assuming almost-everywhere hardness: that no algorithm works *for all sufficiently large input lengths*. That is, the 'infinitely-often' qualifier on the consequent can be *flipped* across the implication to be an 'almost-everywhere' qualifier on the assumption and vice-versa. Thus, the unrealistic 'infinitely-often' notion of tractability can be dropped by slightly strengthening the assumption to the (as argued in Remark

53) realistic 'almost-everywhere' hardness. For *non-uniform* derandomizations this is possible.

Starting with [IW01] and the techniques it introduced, all *uniform* derandomizations have been infinitely-often derandomizations *unable* to flip the io- "across" the hardness vs randomness tradeoff. Our work is the first that is able to do this in the uniform derandomization framework, thus removing the 'infinitely-often' qualifier from our derandomizations.

5.3 Arithmetized Fine-Grained Problems

5.3.1 Arithmetizing k-OV

For hardness amplification, for a given k, we will use a family of polynomials introduced in [BRSV17] to construct fine-grained worst-case to average-case reductions for k-OV_n. The arithmetized k-OV problem has degree only $\mathsf{polylog}(n)$, which enabled classical random self-reducibility of low degree polynomials for worst-case to average-case reductions *inside polynomial time*.

Definition 62 (Arithmetized k-Orthogonal Vectors Family)**.** We arithmetize k-OV_n by the collection of polynomials $\left\{f^k_{n,d,p} : \mathbb{F}_p^{knd} \to \mathbb{F}_p\right\}_{n,d,p\in\mathbb{N}}$ such that the variables are grouped into sets of size nd in the form of a matrix $U_i \in \mathbb{F}_p^{n\times d}$ where the n rows $u_i \in U_i$ are each collections of d variables:

$$f^k_{n,d,p}(U_1,\ldots,U_k) = \sum_{u_1\in U_1,\ldots,u_k\in U_k} \prod_{\ell\in[d]} (1 - u_{1\ell}\cdots u_{k\ell})$$

Lemma 63 (Worst-Case to Average-Case Reduction for counting k-OV [BRSV17])**.** *Let p be the smallest prime number larger than n^k and $d = \lceil \log^2(n) \rceil$. If $f^k_{n,d,p}$ can be computed in $O(n^{k/2+c})$ time for some $c > 0$, then k-OV_n can be **counted** in time $\widetilde{O}(n^{k/2+c})$*

Derandomization from uniform assumptions typically requires two other properties of the assumed hard problem: random self-reducibility and downward self-reducibility. We recall from [BRSV17] that $f^k_{n,d,p}$ satisfies both of these properties. We give a polynomial for k-CLIQUE and

show that it also has the necessary properties in Section 5.3.

Random Self-Reducible.

$f^k_{n,d,p}$ is *random self-reducible* by the following classical lemma [Lip89, FF91] (see [BRSV17] for a proof). Note that degree $\log^2 n$ adds negligibly to the random self-reduction time.

Lemma 64 (Random Self-Reducibility of Polynomials)**.** *If $f : \mathbb{F}_P^N \to \mathbb{F}_P$ is a degree $9 < D < P/12$ polynomial, then there exists a randomized algorithm that takes a circuit $\widehat{C}$ 3/4-approximating f and produces a circuit C exactly computing f, such that the algorithm succeeds with high probability and runs in time poly$(N, D, \log P, |\widehat{C}|)$.*

Downward Self-Reducible.

We will show that $f^k_{n,d,p}$ is downward self-reducible in the sense that, if we we have a way to produce an oracle for $f^k_{\sqrt{n},d,p}$, we can quickly compute $f^k_{n,d,p}$ with it. Compare this to downward self-reducibility going from input size n to $n-1$ in previous uniform derandomizations. We exploit our more dramatic shrinkage and parallelism to later give an almost-everywhere derandomization, instead of an infinitely-often one.

Lemma 65. *If there exists an algorithm A that, on input 1^n, outputs a circuit C computing $f^k_{\sqrt{n},d,p}$, then there exists an algorithm that computes $f^k_{n,d,p}$ in time $O(n^{k/2}|C| + \mathsf{TIME}(A))$.*

Proof. Using A, we print a circuit C computing $f^k_{\sqrt{n},d,p}$ in time $\mathsf{TIME}(A)$. To solve an instance of $f^k_{n,d,p}$, we break up its input as follows.

Intuitively, we break each U_i into $\sqrt{n}$ chunks of $\sqrt{n}$ rows each which partitions the sum of $f^k_{n,d,p}$ into $\sqrt{n}^k$ sub-summands. More formally, for $j \in \left[\sqrt{n}\right]$ let $U_i^j \in \mathbb{F}^{\sqrt{n}\times d}$ be the submatrix of U_i consisting of just the $((j-1)\sqrt{n}+1)^{th}$, $((j-1)\sqrt{n}+2)^{th}, \ldots, ((j-1)\sqrt{n}+\sqrt{n})^{th}$ rows.

Now we can feed $U_1^{j_1}, U_2^{j_2}, \ldots, U_k^{j_k}$ as input to C for all $j_1, j_2, \ldots, j_k \in \left[\sqrt{n}\right]$ and sum the results that C gives. This will give the correct answer by inspection and makes $\sqrt{n}^k$ calls to C. □

5.3.2 Arithmetizing k-CLIQUE

We now construct a family of polynomials from k-CLIQUE that will have the same downward self-reduciblity and random self-reducibility properties as the k-OV polynomials above. Under the weak k-CLIQUE conjecture, these polynomials will be useful for uniform derandomization. We will also discuss how our results hold under an even *weaker* version of the k-CLIQUE conjecture. This polynomial was independently constructed in [GR18]. We first define the k-partite k-CLIQUE problem.

Definition 66 (k-partite k-CLIQUE problem)**.** For an integer $k \geq 3$, the k-CLIQUE$_n$ problem is to, given k graphs on n nodes such that all the edges are only *between* the graphs, decide if there is a k-CLIQUE among them. The input is given as $\binom{k}{2}$ biadjacency matrices between the k graphs.

The k-partite k-CLIQUE problem is fine-grained equivalent to the common k-CLIQUE problem on one graph (i.e. they reduce to each other in $O(n^2)$ time). We now introduce a polynomial that can count k-CLIQUE's.

Definition 67 (Arithmetized k-Clique Family)**.** For a given k, consider the family of polynomials $\left\{g^k_{n,p} : (\mathbb{F}_p^{n\times n})^{\binom{k}{2}} \to \mathbb{F}_p\right\}_{n,p\in\mathbb{N}}$. Overloading notation, let $\binom{k}{2} = \{(i,j) : 1 \leq i < j \leq k\}$. Then,

$$g^k_{n,p}(A^{(1,2)}, A^{(1,3)}, \ldots, A^{(k-1,k)}) = \sum_{v_1,\ldots,v_k\in[n]} \prod_{(i,j)\in\binom{k}{2}} A^{(i,j)}_{v_i,v_j}$$

Note that boolean input corresponds to biadjacency matrices that comprise a k-partite k-CLIQUE instance and that the polynomial counts the number of k-CLIQUE's so long as p is a prime larger than n^k. Now, instead of following the technique of [BRSV17] to find such a prime in time $\widetilde{O}(n^{k/2+c})$ for any $c > 0$, we can use the randomness in our contrapositive derandomization arguments to choose many random primes so that an evaluation of the polynomial over each prime will be able reconstruct the count via the Chinese Remainder Theorem. Since choosing random primes is much faster than finding the single large prime, this along with the following remark will allow us to use a weaker k-CLIQUE conjecture for derandomization.

We relate the worst-case hardness of evaluating these polynomials to the worst-case hardness of k-CLIQUE$_n$ with the following lemma.

Lemma 68. *Let p being the smallest prime number larger than n^k. If $g^k_{n,p}$ can be computed in $O(n^{k/2+c})$ time for some $c > 0$, then k-CLIQUE$_n$ can be **counted** in time $\widetilde{O}(n^{k/2+c})$.*

Remark 69. *The polynomial $g^k_{n,p}$ is guaranteed to be $n^{\omega k/3-o(1)}$ hard under k-CLIQUE's hardness for k a multiple of three but a naïve evaluation takes $\widetilde{O}(n^k)$ time. However, interpreting each matrix of field elements as the biadjacency matrix of a **weighted** graph, the polynomial can be evaluated by the methods of [NP85] as shown in [Lin18]. This faster evaluation solves an open problem of [BRSV17] of finding a polynomial whose computability is **tight** to its hardness for k-CLIQUE. Importantly, this method will speed up the oracle calls in our downward self-reduction in our derandomization arugments, allowing for a weaker k-CLIQUE conjecture.*

We now show that $g^k_{n,p}$ is random self-reducible and downward self-reducible as required by the uniform hardness-to-randomness machienary. Random self-reducibility is automatic as with $f^k_{n,d,p}$ from Lemma 64 (note that our degree is the constant $\binom{k}{2}$ and so adds negligibly to the random self-reduction time), and we will show that $g^k_{n,p}$ reduces to $g^k_{n^\delta,p}$ similarly to $f^k_{n,d,p}$ (we choose δ different than 1/2 since we can now evaluate the polynomial quickly enough to make weaker conjectures). Namely, we will show the following lemma.

Lemma 70. *If there exists an algorithm A that, on input 1^n, outputs a circuit C computing $g^k_{n^\delta,p}$, then there exists an algorithm that computes $g^k_{n,p}$ in time $O(n^{(1-\delta)k}|C| + \mathsf{TIME}(A))$, for any $\delta > 0$.*

Proof. Using A, we print a circuit C computing $g^k_{n^\delta,p}$ in time $\mathsf{TIME}(A)$. To solve an instance $A^{(i,j)}$, $(i,j) \in \binom{k}{2}$, of $g^k_{n,p}$, we break it up as follows.

Let $P = \left\{\{(j-1)n^\delta + 1, (j-1)n^\delta + 2, \ldots, (j-1)n^\delta + n^\delta\} : j \in n^{1-\delta}\right\}$ be a partitioning of $[n]$ into $n^{1-\delta}$ sets of size n^δ each. Then we can see $g^k_{n,p}$ breaks into sub-summands as follows.

$$g^k_{n,p}(A^{(1,2)},A^{(1,3)},\ldots,A^{(k-1,k)}) = \sum_{P_1,\ldots,P_k\in P}\left(\sum_{v_1\in P_1,\ldots,v_k\in P_k}\prod_{(i,j)\in\binom{k}{2}} A^{(i,j)}_{v_i,v_j}\right) \tag{5.1}$$

We claim the inner sum can be computed by $g^k_{n^\delta,p}$ if given the right inputs. Say we have $P_1,\ldots,P_k \in P$. We can make new matrices $B^{(i,j)} \in \mathbb{F}_p^{n^\delta\times n^\delta}$, $(i,j)\in\binom{k}{2}$ as new input for $g^k_{n^\delta,p}$ for C to solve for us:

To create $B^{(i,j)}$ we consider P_i and P_j and fill $B^{(i,j)}$'s entries in as $A^{(i,j)}$ restricted to the submatrix on row indices P_i and column indices P_j. By inspection, these $B^{(i,j)}$ passed as input to $g^k_{n^\delta,p}$ will yield the inner summand for $P_1,\ldots,P_k$.

Thus, feeding these inputs to C for all $P_1,\ldots,P_k$ and summing the results that C gives will give the evaluation of $g^k_{n,p}$ on $A^{(i,j)}$, $(i,j)\in\binom{k}{2}$. This takes $n^{(1-\delta)k}$ calls to C. □

Remark 71. *Since constructing circuit C (from broken derandomization) for the above lemma takes time $\widetilde{O}(n^{\delta\omega k/3+c_2})$ time by using the fast/tight evaluation of $g^k_{n^\delta,p}$ from Remark 69, then evaluating $g^k_{n,p}$ using the lemma will take $\widetilde{O}(n^{(1-\delta)k+c_1}+n^{\delta\omega k/3+c_2})$ time in total, for constants c_1 and c_2. Setting $\delta = 3/(\omega+3)$ is optimal and yields an $\widetilde{O}(n^{\frac{\omega}{\omega+3}k+c})$ algorithm for k-CLIQUE for some constant c. Thus the k-CLIQUE conjecture can be made with $\varepsilon_0 > \omega/(\omega+3)$ instead of $\varepsilon > 1/2$.*

5.4 Fine-Grained Derandomization

We will prove our main derandomization results (Theorems 75 and 80) here. Under either the (weak) k-OV or k-CLIQUE conjectures, we derandomize BPP on average, where 'on average' will have two different flavors. Although all techniques apply similarly to k-CLIQUE, for concreteness we will use k-OV throughout this section.

We first show in Section 5.4.1 that if we base pseudorandom generators on $f^k_{n,d,p}$, then an algorithm printing distinguishers for this PRG can be used to count k-OV solutions quickly.

We will then show in Section 5.4.2 how to obtain these distinguisher-printing algorithms if derandomization *doesn't* work on average (for both flavors of "on average"). Thus, a failed derandomization using these PRGs refutes the k-OV conjecture (similarly for k-CLIQUE).

More specifically, in Section 5.4.2 we will show that the amount of randomness needed can be shrunk in polynomial time to only n^α random bits for any constant $\alpha > 0$ and still succeed in deciding the language on average *over any efficient distribution on inputs* (which implies a *fully deterministic* sub-exponential derandomization over all efficient distributions). The second flavor of derandomization will be shown in Section 5.4.2, that we can fully derandomize in *polynomial* time on average over the *uniform* distribution. Both arguments proceed by constructively mapping failure of derandomization to distinguisher circuits.

5.4.1 Counting k-OV from Distinguishers

In this section we show that any algorithm producing a distinguisher for $G^{f^k_{m,d,p}}$ (the generator guaranteed to exist from Lemma 72, using the hard function $f^k_{m,d,p}$) can be used to quickly count k-OV solutions.

First, Lemma 72 follows immediately by combining the distinguisher to predictor algorithm of Lemma 56 with the fact that $f^k_{m,d,p}$ is random self-reducible (Lemma 64).

Lemma 72. *There is a randomized oracle algorithm $A^{f^k_{m,d,p}}$ that takes any circuit D that is a distinguisher for $G^{f^k_{m,d,p}}$ and produces a circuit C exactly computing $f^k_{m,d,p}$, such that A succeeds with high probability and runs in time* $\mathsf{poly}(m, d, \log p, |D|)$

As usual, having an oracle to $f^k_{m,d,p}$, the assumed hard function, is not desirable and our algorithms dependence on it will need to be removed. We stray from the techniques of [IW01] and worst-case uniform derandomization in general, by using the fact that our problems are from the fine-grained world — and thus polynomial-time computable — to simply answer any oracle queries by brute force! This is (in part) how we can remove the 'infinitely-often' qualifier that plagues all previous worst-case uniform derandomizations.

As $f^k_{m,d,p}$ is computable in time $O(m^k \mathsf{poly}(d, \log p))$, we get the following theorem *without* an oracle by running the algorithm guaranteed in Lemma 72 with each oracle call answered by the naïve brute force computation of $f^k_{m,d,p}$.

Lemma 73. *There is a randomized algorithm B that takes any circuit D that is a distinguisher for $G^{f^k_{m,d,p}}$ and produces a circuit C of size $\mathsf{poly}(m, d, \log p, |D|)$ exactly computing $f^k_{m,d,p}$. B succeeds with high probability and runs in time $O(m^k \mathsf{poly}(m, d, \log p, |D|))$.*

Now we show that, if we have an algorithm producing a distinguisher, then we have an algorithm counting k-OV.

Theorem 74 (Distinguishers to k-OV Solvers)**.** *Let p be the smallest prime number larger than n^k and $d = \lceil \log^2(n) \rceil$. If there is an algorithm A that, on input 1^n, outputs a distinguisher D of $\mathsf{poly}(n)$ size for $G^{f^k_{\sqrt{n},d,p}}$, then there exists a randomized algorithm counting k-OV_n that runs in time $O(n^{k/2+c} + \mathsf{TIME}(A))$, where c only depends on $|D|$.*

Proof. Using A, we print a distinguisher circuit D for $G^{f^k_{\sqrt{n},d,p}}$. Then, by Lemma 73, we know there exists a randomized algorithm running in time $O(n^{k/2}\mathsf{poly}(\sqrt{n}, d, \log p, |D|)) = O(n^{k/2+c_1})$ that yields a circuit exactly computing $f^k_{\sqrt{n},d,p}$ of size only $\mathsf{poly}(\sqrt{n}, d, \log p, |D|) = O(n^{c_2})$, where c_1 and c_2 only depend on $|D|$. Thus, by Lemma 65, there exists an algorithm computing $f^k_{n,d,p}$ in time $O(n^{k/2+c_2} + (n^{k/2+c_1} + \mathsf{TIME}(A))) = O(n^{k/2+c} + \mathsf{TIME}(A))$ for $c = \max\{c_1, c_2\}$. Finally, this gives us an algorithm running in time $\widetilde{O}(n^{k/2+c} + \mathsf{TIME}(A))$ to count k-OV_n by Lemma 63. □

5.4.2 Printing Distinguishers from Failed Derandomization

We now show that, if either of two (shown in 5.4.2 and 5.4.2 respectively) types of derandomization fail, we can uniformly print the distinguishers needed in Section 5.4.1 and thus count k-OV solutions.

Randomness-Reduced Heuristics Over Any Efficient Distribution

Our first main result in derandomizing BPP is to reduce the amount of randomness required to arbitrary inverse-polynomial quantities, over any efficient distribution of inputs. This simulation trades time for reduced randomness under fine-grained hardness assumptions. More precisely, the fewer coins we want to use, the larger k we must select for building a PRG based on k-OV, the longer the runtime of the derandomization. This is the sense in which we have a fine-grained randomness-reduced simulation of BPP: given a desired inverse-polynomial coin-bound, our argument produces concrete polynomial time-bounds on the resulting simulation.

Theorem 75. *If the weak k-OV conjecture holds almost everywhere, then, for all polynomially samplable ensembles μ and for all constants $\alpha > 0$,*

$$(\mathsf{BPP}, \mu) \subseteq \mathrm{HeurBPP}_{[n^{\alpha}]}$$

Thus, for any efficient distribution over inputs that nature might be drawing from and for any inverse polynomial error rate we specify, we can simulate BPP using only n^{α} random bits for any constant $\alpha > 0$ we want. By brute-forcing over all random bits and taking the majority answer over this randomness-reduced computation we can always trivially create a fully deterministic simulation to achieve the following more traditional-looking derandomization result.

Corollary 76. *If the weak k-OV conjecture holds almost everywhere, then, for all polynomially samplable ensembles μ,*

$$(\mathsf{BPP}, \mu) \subseteq \mathrm{HeurSUBEXP}$$

Remark 77. *While we are able to remove the infinitely-often qualifier from our derandomization, note that for each efficient distribution of inputs and each inverse polynomial error rate we are guaranteed to have a derandomizing algorithm, whereas [IW01] is able to achieve a **single** derandomizing algorithm that works for each efficient distribution . Nonetheless, our result*

is strictly stronger as it implies, for instance, $\mathsf{EXP} \neq \mathsf{BPP}$ *(shown in Section 5.5) which is the assumption used to achieve the derandomization of [IW01].*

In contrast to typical full derandomizations which brute-forces all seeds to a pseudorandom generator and take majority answer, we now show that choosing a *single random seed* and using the generator's output as our randomness yields randomness-reduced simulations so long as the generator is efficient enough. Typically, the generator is not fast enough for this application; 'quick' complexity-theoretic PRGs are usually given exponential time in their seed length as they construct pseudorandom strings via queries to problems that have *exponential* hardness. But when life gives you uniform random bits, you can make derandomization lemonade in a "single shot."

Definition 78 (Randomness-Reduced Simulations)**.** Let $A : \{0,1\}^N \times \{0,1\}^{N^\ell} \to \{0,1\}$ be a randomized algorithm that uses N^ℓ random bits and let $G : \{0,1\}^{N^\alpha} \to \{0,1\}^{N^\ell}$ be a function. Then for constant $\alpha > 0$, define the randomness-reduced simulation to be a randomized algorithm $B : \{0,1\}^N \times \{0,1\}^{N^\alpha} \to \{0,1\}$ using only N^α random bits as $B(x,r) = A(x,G(r))$. This is an alternative canonical derandomization.

We now show that if B does not work as a randomness-reduced heuristic, we can uniformly print a distinguisher for the function G.

Lemma 79 (Failed Randomness-Reduction to Distinguishers)**.** *Let A, B, and G be as in Definition 78 such that for language* $\mathscr{L} : \{0,1\}^N \to \{0,1\}$*,*

$$\Pr_{r \sim \mathscr{U}_{N^\ell}} [A(x,r) \neq \mathscr{L}(x)] \leq 1/10$$

That is, that A as a good randomized algorithm deciding $\mathscr{L}$ *for all* $x \in \{0,1\}^N$*. Yet, also assume that, for* μ *samplable in time* N^{a_1} *and* $\delta(n) = 1/N^{a_2}$*, it holds that*

$$\Pr_{x \sim \mu_N} \left[\Pr_{r \sim \mathscr{U}_{N^\alpha}} [B(x,r) \neq \mathscr{L}(x)] > 1/3 \right] \geq \delta(N)$$

That is, B as a (randomness-reduced) randomized algorithm does not decide $\mathscr{L}$ on average over μ. Then $1^N \mapsto \mathsf{DIS}(G, 1/5)$ is in randomized time N^c $\mathsf{TIME}(G)$ for c depending on a_1 and a_2.

Proof. Assume the antecedents of Lemma 79. This means that, with probability at least $\delta(N)$, choosing x from μ will result in an x such that

$$\Pr_{r \sim \mathscr{U}_{N^\alpha}}[B(x,r) \neq \mathscr{L}(x)] > 1/3$$

If we can find an $x' \in \{0,1\}^N$ that induces this "bad" performance of B on $\mathscr{L}$, the test $T(r) = A(x',r)$ will be in $\mathsf{DIS}(G, 1/5)$. That is, since x' makes B perform poorly while A still performs well on *all* x's and since $B(x',z) = A(x', G(z))$, the distinguishing gap of T is

$$\left| \Pr_{r \sim \mathscr{U}_{N^\ell}}[T(r)] - \Pr_{z \sim \mathscr{U}_{N^\alpha}}[T(G(z))] \right| = \left| \Pr_{r \sim \mathscr{U}_{N^\ell}}[A(x',r)] - \Pr_{z \sim \mathscr{U}_{N^\alpha}}[B(x', G(z))] \right| > |1/10 - 1/3| > 1/5$$

To find such an x', we simply sample-and-test as in [IW01]: sample $\widetilde{O}(1/\delta(N))$ possible $x_i \in \{0,1\}^N$'s from μ and, for each of them, define the test $T_i(r) = A(x_i, r)$. For each T_i, in polynomial time we can estimate the fraction of $r \in \{0,1\}^{N^\ell}$ that T_i accepts on and, by making calls to G, we can estimate the fraction of $G(z)$ for $z \in \{0,1\}^{N^\alpha}$ that T_i accepts on in polynomial time times $\mathsf{TIME}(G)$. Thus we can estimate the distinguishing gap for each T_i.

With high probability one of these T_i is a 1/5-distinguisher for G and our estimation of its distinguishing gap reveals it. Print this circuit. □

Randomness-Reduced Simulations from k-OV.

To finish defining a randomness-reduced simulation, we need to use a specific generator G that, for input length N, stretches N^α coins to N^ℓ pseudo-random bits. Thus, consider the family of simulations B_k using the standard generators $G^{f^k_{\sqrt{n},d,p}}$ of Lemma 56 that map $\sqrt{n}^s$ bits to $\sqrt{n}^b$ bits, for some fixed s and any b we choose, using $f^k_{\sqrt{n},d,p}$ as our hard function, for $d = \log^2 n$ and p the smallest prime number larger than n^k. Set $b = s\ell/\alpha$ and $\sqrt{n} = N^{\alpha/s}$. Note that $\mathsf{TIME}(G^{f^k_{\sqrt{n},d,p}}) = \mathsf{poly}(N)\, n^{k/2} = \mathsf{poly}(N)$ by naïvely evaluating $f^k_{\sqrt{n},d,p}$ at each oracle call,

giving an efficient randomness-reduced simulation. Further, since $N = \mathsf{poly}(n)$, $\mathsf{TIME}(G^{f^k_{\sqrt{n},d,p}})$ also equals $n^{k/2+c}$ for some constant c not depending on k (this will be useful in quickly counting k-OV_n using downward self-reducibility in the following proof). Thus, given an N^ℓ-coin machine A, we have the N^α-coin machine $B_k(x,r) = A\left(x, G^{f^k_{\sqrt{n},d,p}}(r)\right)$.

We now prove our main Theorem 75 using this simulation and the above lemma.

Proof of Theorem 75. We proceed towards contradiction. Assume that the weak k-OV_n conjecture holds for all but finitely many input lengths, where $\varepsilon_0 = 1/2 + \gamma$ for some constant $\gamma > 0$, but that there exists $\mathscr{L} \in \mathsf{BPP}$, a polynomially samplable distribution μ, constant α, and an inverse polynomial function $\delta(N)$ such that *any* polynomial-time randomness-reduced algorithm with coin bound N^α fails in deciding $\mathscr{L}$ on average over μ within $\delta(N)$ error for infinitely many input lengths N.

Since $\mathscr{L} \in \mathsf{BPP}$ there is a randomized algorithm A deciding $\mathscr{L}$ with probability of error at most 1/10 over its randomness yet, since *any* polynomial-time randomness-reduced algorithm fails to decide $\mathscr{L}$ on average, B_k, the randomness-reduced simulation of A described above, fails on average infinitely often, for *any* constant k. Thus, the antecedents of Lemma 79 are satisfied and we can uniformly print $D \in \mathsf{DIS}(G^{f^k_{\sqrt{n},d,p}}, 1/5)$ in time $n^{c_1}\ \mathsf{TIME}\left(G^{f^k_{\sqrt{n},d,p}}\right) = n^{c_1}\ n^{c_2} n^{k/2}$.

This uniform printing of D allows us to apply Theorem 74 to count k-OV_n in time $O(n^{k/2+c_3} + n^{k/2+c_1+c_2}) = O(n^{k/2+c}) = O(n^{\left(\frac{1}{2}+\frac{c}{k}\right)k})$ for *any* k, where $c = \max\{c_1 + c_2, c_3\}$. Setting k such that $\frac{c}{k} < \gamma$ yields our contradictions: on the infinitely many input lengths where B_k fails to derandomize $\mathscr{L}$, the algorithm counts k-OV faster than $n^{\varepsilon_0 k}$ time.

□

Fast Heuristics for BPP Over the Uniform Distribution

Here we present our second flavor of derandomization: a fully deterministic heuristic for BPP over inputs sampled according to the uniform distribution.

Theorem 80. *If the weak k-OV conjecture holds almost everywhere, then*

$$(\mathsf{BPP}, \mathscr{U}) \subseteq \mathsf{HeurP}$$

This strictly improves previous uniform derandomizations over the uniform distribution. Specifically, [GW02] can be seen to achieve our derandomization identically from a worst-case uniform assumption if combined with techniques from [KvMS12] *except only on infinitely many input lengths*.

We proceed by showing that if a PRG fails to give a good heuristic for BPP over the uniform distribution, a seed-aware distinguisher for the PRG can be produced uniformly and efficiently, which can then be used to count k-OV solutions quickly using Theorem 74.

Definition 81 (Input-As-Seed Heuristics)**.** Let $A : \{0,1\}^N \times \{0,1\}^{N^\ell} \to \{0,1\}$ be a polynomial-time randomized algorithm using N^ℓ random bits. Let $G : \{0,1\}^N \to \{0,1\}^{N^\ell}$ be a deterministic function. Define the heuristic $B : \{0,1\}^N \to \{0,1\}$ that uses its input as G's seed as $B(x) = A(x, G(x))$.

Now recall the Main Lemma of [KvMS12] giving the consequences of failed heuristics in the *non-uniform* setting:

Lemma 82 (Failed Heuristics $\implies$ Distinguishers)**.** *Let $A : \{0,1\}^N \times \{0,1\}^{N^\ell} \to \{0,1\}$ and $\mathscr{L} : \{0,1\}^N \to \{0,1\}$ be functions such that*

$$\Pr_{x \sim \mathscr{U}_N, r \sim \mathscr{U}_{N^\ell}} [A(x,r) \neq \mathscr{L}(x)] \leq \rho$$

*Let B be the seed-as-input heuristic for A using function G. Then, if B does **not** succeed on a $(3\rho + \varepsilon)$ fraction of the inputs of a given length, there **exists** an $r' \in \{0,1\}^{N^\ell}$ such that the test $T_{r'}(x,r) = A(x,r) \oplus A(x,r')$ is in $\mathsf{DIS}(G, \varepsilon)$.*

The proof of the above lemma uses *non-uniformity* to obtain a good r' for distinguishing, but we can instead *uniformly* obtain good strings r' via a sample-and-test procedure. There is

some loss in the accuracy of the resulting simulation, but this can be made an arbitrarily small inverse polynomial via standard error reduction.

Intuitively, if B is a *bad* heuristic for $\mathscr{L}$, then we could use $B(x) = A(x, G(x))$ as a seed-aware distinguisher for G by comparing $B(x)$ to $\mathscr{L}(x)$. Unfortunately we cannot afford to print distinguishers with $\mathscr{L}$-oracles. But since we are guaranteed that A is a *good* heuristic for $\mathscr{L}$, we can obtain a deterministic circuit close to $\mathscr{L}$ from A, by fixing a string of good random bits r'. If we compare $B(x)$ and the fixed-coin algorithm $A(x, r')$, they will also tend to disagree, giving the necessary distinguishing gap. We can find and fix a good r' uniformly by showing that the set of good random strings is dense, and then sampling and testing for goodness. This sample-and-test process incurs only a constant-factor overhead — a small price to pay for uniformity. Formally:

Lemma 83 (Failed Heuristics $\Longrightarrow$ Uniform Distinguishers)**.** *Let A, $\mathscr{L}$, G, and B be as in Lemma 82 above. Then, if B does **not** succeed on a $(5\rho + \varepsilon)$ fraction of the inputs of a given length, the map $1^N \mapsto \mathsf{DIS}(G, \varepsilon)$ is uniform and in randomized polynomial time, for infinitely many N.*

Proof. By the assumption that A succeeds on a ρ fraction of input-coin pairs, we know that *many* fixings of the coins of A produce an algorithm close to $\mathscr{L}$. We can thus obtain in polynomial time and with high probability a string $r' \in \{0,1\}^{N^\ell}$ such that

$$\Pr_{x \sim \mathscr{U}_N} [A(x, r') \neq \mathscr{L}(x)] \leq 2\rho$$

by repeatedly sampling and testing such fixings of A's coins. This is the first of several "constructive averaging arguments" used here. As above, define the tests,

$$T_{r'}(x, r) = A(x, r) \oplus A(x, r')$$

which output '1' when, for input x, A on fixed coins r' disagrees with A run on input coins r. To output a distinguisher, we simply find an appropriate string r' and then print the circuit $T_{r'}$. This

only takes polynomial time. To show that with high probability $T_{r'} \in \mathsf{DIS}(G, \varepsilon)$, consider two cases.

1. $T(x, G(x))$: We have by definition of $B = A(x, G(x))$ and the assumption that B is *not* a good heuristic that $A(x, G(x))$ will tend to disagree with $\mathscr{L}$. So $A(x, G(x))$ will also tend to disagree with $A(x, r')$. Thus, the test will be biased towards printing 1 when run with generator output.

2. $T(x, \mathscr{U}_{N^\ell})$: The algorithm $A(x, \mathscr{U}_{N^\ell})$ will tend to agree with $A(x, r')$ by our constructive averaging above. So the test will tend to output 0 when run with true random bits.

These cases correspond to the first and second terms of the distinguishing gap equation (Definition 55), which we substitute the test into below:

$$\left| \underbrace{\Pr_{x \sim \mathscr{U}_N} \left[A(x, G(x)) \neq A(x, r') \right]}_{(i)} - \underbrace{\Pr_{x \sim \mathscr{U}_N, R \sim \mathscr{U}_{N^\ell}} \left[A(x, r) \neq A(x, r') \right]}_{(ii)} \right|$$

An upper bound on term (ii) is straightforward by union bound and Fréchet inequality: it is at most 3ρ. For term (ii), observe that we have the following bounds on relative Hamming distance from $A(x, G(x))$ to $\mathscr{L}$, and from $A(x, r')$ to $\mathscr{L}$ at the nth slice:

$$d(A(x, G(x)),\ \mathscr{L}) \geq 5\rho + \varepsilon \qquad \text{by assumption on B}$$

$$d(A(x, r'),\ \mathscr{L}) \leq 2\rho \qquad \text{by constructive averaging}$$

So, by triangle inequality, we obtain a lower bound of $3\rho + \varepsilon$ on the first term of the distinguisher equation (with high probability). Therefore, if our constructive averaging succeeds in finding a good r', we have $T_{r'} \in \mathsf{DIS}(G, \varepsilon)$. The time to print $T_{r'}$ is just the polynomial time to find a good r' by repeatedly sampling and running A to test, plus the time to print A as a circuit. Thus the

size of the distinguisher printed is only $\widetilde{O}(\mathsf{TIME}(A))$ — it does not depend on the runtime of the constructive averaging argument. □

Fully Deterministic Heuristics from k-OV.

Here we specify a family of heuristics B_k, by fixing the generator G, that stretches a seed of length N to N^ℓ, as the generators $G^{f^k_{\sqrt{n},d,p}}$ of Lemma 56. These map $\sqrt{n}^s$ bits to $\sqrt{n}^b$ bits, for some fixed s and any b we choose, using $f^k_{\sqrt{n},d,p}$, for $d = \log^2 n$ and p the smallest prime number larger than n^k. Set $b = s\ell$ and $\sqrt{n} = N^{1/s}$. All comments about the runtime of the randomness-reduced heuristic in Section 5.4.2 also apply to this fully deterministic heuristic. Thus, given an N^ℓ-coin machine A, we define the *deterministic* machine $B_k(x) = A\left(x, G^{f^k_{\sqrt{n},d,p}}(x)\right)$.

Note that $\mathsf{TIME}(G^{f^k_{\sqrt{n},d,p}}) = \mathsf{poly}(N)\ n^{k/2} = \mathsf{poly}(N)$ by naïvely evaluating $f^k_{\sqrt{n},d,p}$ at each oracle call, giving an efficient randomness-reduced simulation. Further, observe that $N = \mathsf{poly}(n)$ so that $\mathsf{TIME}(G^{f^k_{\sqrt{n},d,p}})$ also equals $n^{k/2+c}$ for some constant c not depending on k (this will be useful in quickly counting k-OV_n using downward self-reducibility in the following proof). Given $\mathscr{L} \in \mathsf{BPP}$ and A the N^ℓ-coin machine for $\mathscr{L}$, define $B_k(x,r) = A(x, G^{f^k_{\sqrt{n},d,p}}(r))$

We now prove our main Theorem 80 using this simulation and the above lemma.

Proof of Theorem 80. We proceed by contradiction. Assume that the weak k-OV_n conjecture holds for all but finitely many input lengths, where $\varepsilon_0 = 1/2 + \gamma$ for some constant $\gamma > 0$, but that there exists $\mathscr{L} \in \mathsf{BPP}$ and an inverse polynomial function $\delta(N)$ such that *any* polynomial-time deterministic algorithm fails in deciding $\mathscr{L}$ on average over μ within $\delta(N)$ error for infinitely many input lengths N.

Namely, since $\mathscr{L} \in \mathsf{BPP}$ there is a randomized algorithm A deciding $\mathscr{L}$ with probability of failing over its coins at most $\rho(N)$ for any inverse polynomial (by standard error reduction), yet it must be the case that our B_k simulation of A described above fails on average infinitely often as well, for *any* constant k. Thus, choose inverse polynomial $\varepsilon(N)$ and $\rho(N)$ such that $5\rho + \varepsilon \leq \delta(N)$ and this failure satisfies the preconditions of Lemma 83 and thus we can uniformly print $D \in \mathsf{DIS}(G^{f^k_{\sqrt{n},d,p}}, \varepsilon)$ in time $n^{k/2+c_1}$.

Again this allows us to apply Theorem 74, which counts k-OV in time $O(n^{k/2+c_2} + n^{k/2+c_1}) = O(n^{\left(\frac{1}{2}+\frac{c}{k}\right)k})$ for *any* k, where $c = \max\{c_1, c_2\}$. Setting k such that $\frac{c}{k} < \gamma$ yields our contradiction: on the infinitely many input lengths where B fails to derandomize $\mathscr{L}$, the algorithm counts k-OV_n faster than $n^{\varepsilon_0 k}$ time. □

5.5 Heuristics Imply Separations

We now show that, if a deterministic simulation of BPP succeeds infinitely-often and on average, then BPP doesn't contain any deterministic time class larger than P. This is strong limitation on the power of BPP from a strong assumption. Compare this to Corollary 7 of [IW01] (reproduced below) which starts from a much weaker assumption but gets a corrospondingly weaker separation. This pair of results reinforces our theme of obtaining high-end analogs to [IW01].

Theorem 84 (High-end Heuristics for BPP $\implies$ Uniform Separation)**.** *Suppose* $(\mathsf{BPP}, \mathscr{U}) \subseteq \mathsf{io\text{-}Heur}_{1/3}\mathsf{P}$. *Then, for all* $t(n) = n^{\omega(1)}$ *time-constructible:*

$$\mathsf{DTIME}[t(n)] \nsubseteq \mathsf{BPP}$$

Theorem 85 (Low-End Heuristics for BPP $\implies$ Uniform Separation [IW01])**.** *If* $\mathsf{BPP} \subset \mathsf{io\text{-}Heur}_{1/3}\mathsf{TIME}[2^{o(n)}]/o(n)$ *then* $\mathsf{BPP} \neq \mathsf{EXP}$

Intuitively, suppose we were able to fully derandomize, ie, prove $\mathsf{BPP} \subseteq \mathsf{P}$. Then the determinisic time-hierarchy theorem would imply $\mathsf{DTIME}[n^{\omega(1)}] \nsubseteq \mathsf{BPP}$. Thus, proving a time-hierarchy theorem that is robust to the io- and Heur qualifiers suffices to conclude $\mathsf{DTIME}[n^{\omega(1)}] \nsubseteq \mathsf{BPP}$ from a $(\mathsf{BPP}, \mathscr{U}) \subseteq \mathsf{io\text{-}HeurP}$ derandomization. We expand ideas from [IW01] to prove the following sufficient lemma.

Lemma 86 (Robust Time Hierarchy Theorem)**.** *For all time-constructible $t(n)$:*

$$\text{io-Heur}_{1/3}\text{DTIME}[t(n)] \subset \text{DTIME}[t(n)^3]$$

Proof. We give a deterministic machine M that runs in time $t(n)^3$ but whose language is not in $\text{io-Heur}_{1/3}\text{DTIME}[t(n)]$:

Let $\ell(n) = \log t(n)$. On input $x = u||v$, where u is the first $n - \ell(n)$ bits and v is the last $\ell(n)$ bits, M simulates all Turing machines of description length $(1/2)\ell(n)$ on every n-bit input that begins with u for $t(n)$ steps. This creates $2^{.5\ell(n)} = \sqrt{t(n)}$ strings of length $t(n)$ which are the truth tables of each $.5\ell(n)$-size Turing machines that runs in $t(n)$ steps on all of the $2^{\ell(n)} = t(n)$ inputs that begin with u.

It is easy to see with Chebyshev that a random string of length $t(n)$ agrees with any fixed $t(n)$-length string on at least $2/3$ of its values only with probability $1/O(t(n))$. Since there are only $\sqrt{t(n)} < O(t(n))$ strings that are our truth tables though, by union bound there must exist a $t(n)$-length string that disagrees with *all* of our truth tables on at least 1/3 of each of their values. Of course this string might have high complexity to generate but, since our analysis here only involved a Chebyshev bound, a pairwise independent hash family will fool this analysis and yeild the same conclusion.

Namely, considering a random string from the pairwise independent hash family $\mathscr{H} = \{\langle r, \cdot \rangle : r \in \{0,1\}^{\ell(n)}\}$ is sufficient for the analysis and so we have that there must exist a specific $\langle r_u, \cdot \rangle$ that disagrees with all of the truth tables on at least 1/3 of their values. Thus, since $\mathscr{H}$ is relatively small and its functions are easy to compute, M can find that r_u by brute force and outputs its final binary value as $\langle r_u, v \rangle$.

Thus, this whole process of M on $x = u||v$ of simulating $\sqrt{t(n)}$ Turing Machines on $t(n)$ inputs for $t(n)$ time and checking all $t(n)$ r's for the one that fools all of the truth tables adequately enough takes at most $t(n)^3$ time for large enough n. However, by construction, all time $t(n)$ Turing machines fail in deciding this language on at least 1/3 of its inputs for all

sufficiently large n.

□

5.6 Open Questions

- We derandomize under hardness conjectures about two of four 'key' problems in fine-grained complexity: k-OV and k-CLIQUE. What about k-SUM and APSP? APSP doesn't seem to have a natural hierarchy and so doesn't fit our framework (although it does reduce to ZERO-TRIANGLE which generalizes to ZERO-k-CLIQUE and should easily work in our framework using polynomials similar to those in [BRSV17]). k-SUM however is actually computable in $O(n^{\lceil k/2 \rceil})$ time and so our downward self-reducibility techniques are not fast enough to break this conjecture in the contrapositive. The clearest path we see to getting derandomization without reintroducing the io- qualifier is to find a polynomial for k-SUM that is also computable in $\widetilde{O}(n^{\lceil k/2 \rceil})$ time (unlike the one found in [BRSV17]).

- Our derandomizations hold under (randomized) SETH, since SETH implies the k-OV conjecture. Can a better derandomization be obtained directly from SETH, the stronger assumption? A stumbling block here is the random self-reduction, an ingredient in all known uniform derandomization techniques: If t-SAT has a straightforward and efficient random-self-reduction, PH collapses [FF93, BT06b]. So derandomizing from SETH directly could require new ideas, or a strange random self-reduction (such as that of [GST07]). An *inefficient* random self-reduction for t-SAT shouldn't collapse PH except to say that t-SAT has a mildly exponential MA proof which is already known to be true [Wil16], although most random self-reductions we know are through arithmetization which seems to always have 'low' degree to the point that such a polynomial would still collapse PH.

- Is a strong "derandomization to hardness" converse possible for these heuristic simulations

of BPP? In Section 5.5, we showed a weak converse: our simulation is impossible without separting $\mathsf{DTIME}[n^{\omega(1)}]$ from BPP. But this is a very different statement from the k-OV or k-CLIQUE conjectures. In [KvMS12], they show that herusitic simulations of BPP with inverse-subexponential error rates imply circuit lower bounds, by generalizing techniques of [KI04]. Do the efficient inverse-polynomial error heuristics we obtain imply any circuit lower bounds?

- For error-free derandomization, [KI04] shows that circuit lower bounds are necessary. For inverse-subexponential error heuristics, [KvMS12] shows that circuit lower bounds are necessary. Is it possible to conclude circuit lower bounds from the almost-everywhere, inverse-polynomial error heuristics that our techniques produce?

- Could we use the connections established here to show a Karp-Lipton type theorem *inside* BPP? What other classical results be ported "down" into the fine-grained structure of P and executed uniformly? Because hardness vs. randomness trade-offs are common tools in structural complexity, we hope that our derandomizations are useful for this broader project.

Chapter 5, in part, is based on the material as it appears in "Marco L. Carmosino, Russell Impagliazzo, and Manuel Sabin. Fine-grained derandomization: From problem-centric to resource-centric complexity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 27:1–27:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018". The dissertation author was the primary investigator and author of this paper.

Chapter 6

Tighter Connections between Derandomization and Circuit Lower Bounds

6.1 Introduction

While randomness has become an indispensable tool for algorithm design, many (though not all) randomized algorithms have later been derandomized, i.e., shown to have equivalent, comparably fast deterministic algorithms. One highly successful method for derandomization of whole classes of algorithms has been the "hardness as randomness" paradigm [AW85, BM84, Yao82, NW94, BFNW93, IW97, STV01, SU05, Uma03]. In this paradigm, problems that are hard for the type of computation to be derandomized are converted into indistinguishable pseudorandom generators (PRGs) for the same class, which then replace the random choices of the randomized algorithm.

It has long been observed that this method seems to require hardness for the nonuniform version of the class (at least for worst-case derandomization — contrast this with the heuristic derandomizations of Chapter REFXX). PRGs are a special case of "black-box" derandomization methods, where the algorithm to be accessed is only modified by replacing the random decisions with deterministic choices. It is relatively straightforward to show that "black-box" derandomization requires a circuit bound against the type of algorithm to be derandomized (see, e.g., [HS82]). Thus, via the hardness as randomness paradigm, strong circuit lower bounds can be proved *equivalent* to universal "black-box" derandomization.

More surprisingly, there are results that show the reverse direction, that derandomization requires strong circuit lower bounds, is true even for non-black-box algorithms [IKW02]. In fact, even derandomizing a specific algorithm, the randomized polynomial identity test (PIT) of Schwartz and Zippel (also discovered by DeMillo and Lipton) [Sch80, Zip79, DL78], would imply strong lower bounds for arithmetic circuits [KI04, JS12, AM11]. However, the proofs of these statements are not direct reductions, but instead go through various complexity class collapses, and the conclusions in one direction are not usually exact matches for the other. So, unlike for "black-box" algorithms, we do not usually get a literal equivalence between a derandomization result and a circuit lower bound (one exception is [JS12]).

In this paper, we tighten the connections between circuit lower bounds and derandomization in several ways, for each of the three types of derandomization: general derandomization of promise-BPP (connected to Boolean circuits), derandomization of PIT over fixed finite fields (connected to arithmetic circuit lower bounds over the same field), and derandomization of PIT over the integers (connected to arithmetic circuit lower bounds over the integers). We show how to make these connections *equivalences*, although at the expense of using somewhat less common versions of complexity classes and a less studied notion of inclusion, the "robustly-often" inclusion introduced by [FS11]. Even for worst-case inclusion, we simplify and strengthen the known connections in several ways.

6.1.1 Our results

Following [JS12], let ml-NE denote the class of multilinear n-variate polynomials $F = \{f_n\}_{n\geq 0}$ over a domain $\mathscr{D}$ (for $\mathscr{D}$ the set of integers or a finite field) whose graph:

$$\{(a_1,\ldots,a_n,f_n(a_1,\ldots,a_n)) \mid n \geq 0,\ a_i \in \mathscr{D},\ 1 \leq i \leq n\}$$

is in the class $\mathsf{NE} = \mathsf{NTIME}[2^{O(n)}]$.

- For finite fields, we show that derandomization of PIT implies that some polynomial

$F \in \mathsf{ml\text{-}NE}$ does not have polynomial sized arithmetic circuits over the same field, nor does any polynomially bounded power of F. This is the first result getting circuit lower bounds from derandomization of a randomized PIT algorithm over fixed *finite* fields[1]. See Theorem 105 on page 172.

- We improve on the advice requirements for known connections between derandomization of PIT over the integers and arithmetic circuit lower bounds over the integers. See Theorem 107 on page 182.

- Impagliazzo et al. [IKW02] showed that (even nondeterministic) polynomial-time derandomization of promise-BPP would imply a Boolean circuit lower bound for NEXP. We strengthen this to a circuit lower bound for the smaller class $\mathsf{NEXP} \cap \mathsf{coNEXP}$. This is the same class where sufficiently strong circuit lower bounds would imply that $\mathsf{promise\text{-}BPP} \subseteq \mathsf{NP}$. See Theorem 109 on page 183.

- [IKW02] also showed that a Boolean circuit lower bound for NEXP would follow from the existence of an NP-computable non-empty property of Boolean functions that is useful for proving superpolynomial circuit lower bounds (in the sense of natural proofs of Razborov and Rudich [RR97]). We also strengthen this to the lower bound for the class $\mathsf{NEXP} \cap \mathsf{coNEXP}$. See Theorem 111 on page 192.

- The two notions of inclusion and hardness most frequently used in complexity are "every length" inclusion and hardness, and "infinitely often" inclusion and hardness. Unfortunately, the negation of everywhere inclusion is only infinitely often hardness, and vice versa. This prevents many of the connections above from being literal equivalences between lower bounds and derandomization. Using a third notion of inclusion, a variant of the "robustly often" inclusion of [FS11], we make all four of the above connections into

[1] Kabanets and Impagliazzo [KI04] prove a related but weaker version of the hardness-to-pseudorandomness result for finite fields: their arithmetic-complexity hardness assumption is not for a polynomial (defined over all extension fields), but rather for the function that *agrees* with the polynomial over a *particular* extension field.

literal equivalences[2].

6.1.2 Overview of Techniques

How can we argue that an upper bound on algorithmic complexity (efficient derandomization) yields a lower bound on non-uniform complexity (circuit lower bounds)? The various results in this area all follow the same general template (see, e.g., [AvM12] for a formal treatment): We assume that we have both a circuit upper bound for some large class such as NEXP, and a general derandomization technique, and work towards contradiction as outlined below.

Simulation: Meyer, Karp and Lipton [KL82] introduced techniques to give consequences of non-uniform (circuit) upper bounds for uniform classes. Using interactive proofs [BM88, Sha92], many of these can be extended to show that if a large complexity class C such as EXP or PSPACE has small circuits, then C also has short (constant-round) interactive proofs (see, e.g., [BFNW93]).

Derandomization: Invoking the generic derandomization technique in the above simulation, we get that C can be simulated only with non-determinism.

Contradict a known lower bound: If C is NEXP itself, as in [IKW02] and [Wil14b], the contradiction comes from some version of the non-deterministic time hierarchy theorem. An alternate method used by [AM11] is to pad up the non-trivial simulation of C in small non-deterministic time to show that some analogous class superpoly-C can be simulated in NEXP. The class superpoly-C will often be strong enough that a lower bound in that class can be shown by direct diagonalization (see, e.g., [Kan82]). By simulation, NEXP will inherit the same lower bound.

If both the circuit upper bound assumed and the derandomization assumed are worst-case,

[2] [JS12] has an alternative method of getting an equivalence for the case of PIT over the *integers*, but at the expense of moving to versions of the classes with *advice*.

then the above template can be filled out to get a variety of trade-offs. However, when one or the other is only assumed to occur for infinitely many lengths of input, care needs to be taken to compare the input sizes where the simulation is possible, the corresponding lengths where derandomization is possible, and what non-worst-case versions of the known lower bounds are true.

6.1.3 Related Work

While the general issue of connections between circuit complexity and derandomization has been the subject of intense research by many people, the papers most directly parallel to this work are [IKW02, KI04, AM11, JS12, KMS12]. Here, we briefly describe the main results and techniques of these five papers, and compare them to our results and techniques.

Promise-BPP.

[IKW02] considers the question of derandomizing promise-BPP. While there are several other results in the paper, the main result for our purposes is an equivalence between a circuit lower bound for NEXP and a "non-trivial" simulation of promise-BPP.

Theorem 87 ([IKW02])**.** $\mathsf{NEXP} \not\subset \mathsf{P/poly}$ *if and only if* $\mathsf{promise\text{-}BPP} \subseteq \cap_{\varepsilon>0}\mathsf{io\text{-}NTIME}[2^{n^\varepsilon}]/n^\varepsilon$.

While this is a strong result, and in fact an equivalence, there are some questions raised by the details in this statement. Since $\mathsf{promise\text{-}BPP} \subseteq \mathsf{P/poly}$, we know that advice can be powerful in this context. How significant is the small amount of advice allowed in the sub-exponential time algorithms for promise-BPP? Is it really necessary or just a by-product of the proof? Can we get a stronger conclusion if we assume a Circuit Acceptance Probability Problem (estimating the acceptance probability of a size-n Boolean circuit to within an additive error of at most $1/n$; *CAPP*) algorithm with no advice? Also, [IKW02] use the "Easy Witness Lemma" to obtain their result, which is itself derived by a somewhat convoluted indirect derandomization argument. Is this use of the Easy Witness Lemma necessary?

Here, we give some answers to these questions. Using arguments parallel to those in [AM11, KMS12] with respect to PIT, we remove the Easy Witness Lemma. Then (as [AM11] did for PIT) we show that *sufficiently strong nondeterministic algorithms for* promise-BPP *(without advice) imply the stronger conclusion that* $(\mathsf{NEXP} \cap \mathsf{coNEXP}) \not\subseteq \mathsf{P/poly}$ (Theorem 109). So there does seem to be a real difference between derandomizations with and without advice.

One use of advice in the original argument is that, if we have a different CAPP algorithm for every $\varepsilon > 0$, then, for each length, the best value of ε and the algorithm that achieves that ε can be both given as advice. To remove this advice, we need to have a *single* algorithm that runs in nondeterministic sub-exponential time, $2^{n^{\varepsilon(n)}}$ for some *computable* $\varepsilon(n) \in o(1)$. This seems an equally natural definition of a problem being computable in sub-exponential time, and we adopt it throughout our paper. We prove some closure properties and normal forms for this notion that might be of independent interest.

Circuit lower bounds and nontrivial useful properties.

Razborov and Rudich [RR97] defined an NP-constructive property useful against P/poly as an NP-computable predicate on 2^n-bit inputs (the truth tables of n-variate Boolean functions) such that, whenever the predicate holds for infinitely many input lengths of a given family of Boolean functions $f = \{f_n\}_{n \geq 0}$, it follows that $f \notin \mathsf{P/poly}$. Call such a property *nontrivial* if there are infinitely many input lengths n such that at least one truth table of size 2^n satisfies the property.

It was shown in [IKW02] that the existence of a nontrivial NP-constructive property useful against P/poly implies that $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$. We strengthen this to the circuit lower bound for $\mathsf{NEXP} \cap \mathsf{coNEXP}$ for nontrivial NP-constructive properties useful against computably superpolynomial circuit size. Moreover, we make this connection into an equivalence, using a variant of the "robustly often" notion of [FS11] (Theorem 111).

Polynomial identity testing over the rationals.

[KI04] extends the connections between derandomization and circuit lower bounds to the arithmetic-complexity setting. In one direction, they show that if both $\mathsf{NEXP} \subseteq \mathsf{P/poly}$ and the permanent function has polynomial-size arithmetic circuits over the rationals, then no nondeterministic sub-exponential time algorithm can solve PIT, even for infinitely many input sizes. In the other direction, either $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$ or the permanent requiring super-polynomial sized circuits gives some nontrivial algorithm for PIT, although not quite an exact converse.

This again raises some questions. Stating the result in the contrapositive, a nontrivial algorithm for PIT would yield either a Boolean circuit lower bound or an arithmetic circuit lower bound. Intuitively, Boolean circuit lower bounds should be more difficult, so can we get a similar result that only mentions arithmetic circuit lower bounds?

This question was addressed in [AM11, KMS12] and [JS12]. First, [AM11] and [KMS12] give an alternate version of the final contradiction step that allows them to avoid the Easy Witness Lemma. This not only simplified the proof, but allowed them to replace the condition that $\mathsf{NEXP} \subset \mathsf{P/poly}$ with the weaker condition that $(\mathsf{NEXP} \cap \mathsf{coNEXP}) \subset \mathsf{P/poly}$. ([KI04] had also shown that $(\mathsf{NEXP} \cap \mathsf{coNEXP}) \subset \mathsf{P/poly}$ and the easiness of the permanent sufficed to show $\mathsf{PIT} \notin \mathsf{NP}$.) [JS12] show that the Boolean and arithmetic lower bounds can be in some sense combined. They show that derandomizing a certain version of PIT, low-PIT, in non-deterministic sub-exponential time is equivalent to proving a super-polynomial lower bound for circuits with restricted degrees on a multi-linear function whose values can be computed in $\mathsf{NEXP}/O(n)$. However, being determined by the correct advice, the functions where the arithmetic lower bound would hold in their result are somewhat non-constructive, and when the advice is incorrect, the corresponding functions might not be defined at all, so they cannot be combined into one universal function for each length.

We show that by using the [AM11, KMS12] type contradiction step, the two lower bounds can be replaced by a *single arithmetic circuit lower bound for a multilinear polynomial definable in* $\mathsf{NEXP} \cap \mathsf{coNEXP}$ (Theorem 107), thus removing this somewhat awkward non-uniformity.

Polynomial identity testing over a fixed finite field.

The polynomial identity testing problem and arithmetic circuit complexity are also important over other fields, such as fixed finite fields. Here one fixes some constant-size finite field $\mathbb{F}$, and then asks for an efficient algorithm to decide if a given arithmetic circuit C over $\mathbb{F}$, defining a low-degree polynomial p (over $\mathbb{F}$ and any finite field extending $\mathbb{F}$), is such that $p \equiv 0$. The latter can be easily decided by a randomized polynomial-time Schwartz-Zippel algorithm evaluating p on random points from a sufficiently large (larger than the degree of p) extension field of $\mathbb{F}$.

Do similar connections between hardness and pseudorandomness hold for fixed finite fields? Before the work here, almost no such connections were known. There are a number of obstacles to directly translating the [KI04] techniques to fixed finite fields. First, in the "derandomization of PIT to arithmetic hardness" direction, [KI04] use the clean algebraic recursive "expansion by minors" definition of the permanent, a problem complete for #P. While the same recursion holds for the permanent over finite fields, permanent over finite fields is not known to be #P complete, or even NP-hard (under deterministic reductions).

Secondly, in the "arithmetic hardness to derandomization of PIT" direction, if the hitting set generator from [KI04] fails using a given f as the hard function over a finite field, it only follows that some power of f had a small arithmetic circuit, not f itself. Though this power of f can be manipulated to obtain an arithmetic circuit that agrees with f over some particular extension field, the polynomial computed by this circuit would not be identically equivalent to f (see Section 6.2.4 for the difference between testing for identity and agreement of polynomials over a finite field). Therefore, a power of f could still have small arithmetic circuits and this would not contradict the hardness assumption used by [KI04] in the hardness to randomness direction. Thus, [KI04] has only a weak hardness-to-randomness result in the case of circuits over finite fields.

We give the first proof that *nondeterministic polynomial identity testing over a fixed finite field yields an arithmetic circuit lower bound* (Theorem 105). We avoid the permanent

by showing a simulation lemma for an even larger class, PSPACE, using a PSPACE-complete function of [TV02] with an algebraic recursive definition. We observe that, not only would a small circuit for this function itself yield the required simulation, but also any circuit for a small power of the function would have a similar consequence. This matches the type of arithmetic lower bounds needed to derandomize polynomial identity testing using the hitting set generator of [KI04]. Thus, the correct connection (over finite fields) is between derandomizing PIT and proving lower bounds on circuits that compute *powers* of polynomials.

Equivalence between circuit lower bounds and derandomization.

While the results above come closer to showing that each of these three derandomization problems are equivalent to a corresponding lower bound, they are not literal equivalences. This is because of the distinction between infinitely often computation and worst-case computation. Making worst-case assumptions about algorithms is necessary to get even infinitely often hardness, but infinitely often hardness only suffices to get algorithms that work infinitely often.

To make versions of our results that are literal equivalences, we consider an intermediate notion between worst-case and infinitely-often computation, robustly-often computation introduced by [JS12]. Informally, an algorithm solves a given problem *robustly often* if there are infinitely many *intervals* of superpolynomially many input lengths where the algorithm is correct on each length in the interval.

While robust inclusion and separation are somewhat non-standard, they are exactly what we need to make the connections between circuit lower bounds and worst-case derandomization into equivalences (Theorems 106, 108, and 110). We feel that this is not a coincidence, and the "robustness" notion is quite natural and can be justified independently by the following considerations: as technology improves in general, all computational elements will improve somewhat. But computational elements will often get more diverse, so that say, the CPU in cell phones will not be improving its computational power as quickly as the top super-computer will. A reasonable model is to assume that there are two different Moore's laws, one for "high-

end" technology and one for "low-end", with different periods of doubling. Thus, high-end and low-end technologies will be polynomially related in their resources, and the intermediate technologies will span the range between them. Thus, it is natural to look at polynomially related ranges of parameters, not just single values. Robust computation does just that, looking at whether computation is possible not just on infinitely many single input lengths, but whether there are infinitely many "technology levels" of unbounded polynomial reach where this computation is possible.

Organization

- §6.2: We cover basic material regarding: arithmetic circuit complexity (Section 6.2.1), calculation of arithmetic functions by Boolean complexity classes (Section 6.2.2), derandomization of PIT (Section 6.2.4), and finally derandomization of promise-BPP (Section 6.2.5).

- §6.3: We introduce the notions of "robust" inclusion and separation that will allow us to obtain equivalences between lower bounds and derandomization. We establish closure properties of robustness that are important for our applications.

- § 6.4: We show how to arithmetize TQBF to get a PSPACE-complete polynomial whose arithmetic circuits can be tested for correctness with a PIT algorithm.

- § 6.5: We give tighter connections between PIT algorithms and arithmetic circuit lower bounds over finite fields.

- § 6.6: We give tighter connections between PIT algorithms and arithmetic circuit lower bounds over the integers.

- § 6.7: We give tighter connections between derandomizing promise-BPP and Boolean circuit lower bounds.

6.2 Definitions & Tools

6.2.1 Arithmetic circuit complexity classes

An arithmetic circuit on n variables is a directed acyclic graph with n in-degree 0 nodes labeled by the variables $x_1, \ldots, x_n$ (inputs), and one out-degree 0 node (output); each in-degree 2 node is labeled by an arithmetic operation $+$ or $*$; each non-input in-degree 0 node is labeled by a constant. The size of an arithmetic circuit is the number of nodes (gates) in the graph. Each arithmetic circuit $C(x_1, \ldots, x_n)$ computes a polynomial in variables $x_1, \ldots, x_n$, which we shall also denote by $C(x_1, \ldots, x_n)$.

We will consider arithmetic circuits both over integers and over finite fields. Over the integers, we shall consider only constant-free circuits, i.e., those circuits where all constant labels are restricted to the set $\{0, 1, -1\}$.

For a finite field $\mathbb{F}$, define $\mathsf{ASize}_{\mathbb{F}}[s(n)]$ to be the class of all families of n-variate polynomials $\{f_n\}$ over $\mathbb{F}$ such that, for all sufficiently large n, the polynomial f_n is computed by some arithmetic circuit C_n of size at most $s(n)$ over $\mathbb{F}$ (i.e., the polynomials $f_n(x_1, \ldots, x_n)$ and $C_n(x_1, \ldots, x_n)$ are formally the same when viewed as the sums of monomials). Over the integers, the class $\mathsf{ASize}_{\mathbb{Z}}[s(n)]$ is defined analogously.

We denote by $\mathsf{ASizeDeg}_{\mathbb{F}}[s(n)]$ the subclass of $\mathsf{ASize}_{\mathbb{F}}[s(n)]$ of families of n-variate polynomials f_n that have degree at most $s(n)$. Over the integers, we denote by $\mathsf{ASizeDeg}_{\mathbb{Z}}[s(n)]$ the restricted class of polynomial families $\{f_n\}$ that have *formal degree* at most $s(n)$. Recall that the formal degree is defined inductively as follows: for input gates, regardless of their label, the formal degree is 1; an addition gate has the formal degree equal to the maximum of the formal degree of its input gates; a multiplication gate has the formal degree equal to the sum of the formal degrees of its input gates. The bounded formal degree is useful as it allows us to bound the bit-length of any integers computed by a given arithmetic circuit. This is an appealing way to ensure that it is feasible to evaluate the circuit.

"Easy" polynomials.

Over the integers, we shall consider a polynomial f "easy" if some constant multiple $c \cdot f$ is computable by a "small" arithmetic circuit. More formally, we define $\mathsf{ASizeDegMul}[s(n)]$ as the class of polynomial families $\{f_n\}$ over $\mathbb{Z}$ of degree at most $s(n)$ such that, for some function $g : \mathbb{N} \to \mathbb{N}$ we have $\{g(n) \times f_n\} \in \mathsf{ASizeDeg}_{\mathbb{Z}}[s(n)]$ and g is computed by a family of variable-free $\mathsf{ASizeDeg}_{\mathbb{Z}}[s(n)]$ circuits. Thus, for each input length n, we could compute a *different* constant multiple of f. $\mathsf{ASizeDegMul}$ is equivalent to the class $ASIZEDEG'$ of [JS12], we only formulate it differently to ease the description of our identity tests.

Over finite fields, we shall consider a polynomial f "easy" if some bounded power f^d of f is computable by a "small" arithmetic circuit. More formally, over a finite field $\mathbb{F}$, we define the class $\mathsf{ASizeDegPow}_{\mathbb{F}}[s(n)]$ as the class of polynomial families $\{f_n\}$ over $\mathbb{F}$ of degree at most $s(n)$ such that, for some function $d : \mathbb{N} \to \mathbb{N}$ with $d(n) \leq s(n)$, we have $\{f_n^{d(n)}\} \in \mathsf{ASizeDeg}_{\mathbb{F}}[s(n)]$.

The definition of $\mathsf{ASizeDegPow}_{\mathbb{F}}$ does not change if we require that the function $d(n) \leq s(n)$ be such that each $d(n) = p^{k_n}$ for some $k_n \in \mathbb{N}$ where p is the prime characteristic of the finite field $\mathbb{F}$. This is a convienent normal form for our hardness to randomness tradeoffs. Below, we state and prove it formally.

Lemma 88. *Let $\mathbb{F}$ be a finite field of characteristic p. If $f \in \mathsf{ASizeDegPow}_{\mathbb{F}}[s(n)]$, then there exists a function $k : \mathbb{N} \to \mathbb{N}$ such that $p^{k(n)} \leq s(n)$ and $\{f_n^{p^{k(n)}}\} \in \mathsf{ASizeDeg}_{\mathbb{F}}[\mathsf{poly}(s(n))]$.*

Proof. Let $d(n) \leq s(n)$ be such that $\{f_n^{d(n)}\} \in \mathsf{ASizeDeg}_{\mathbb{F}}[s(n)]$. For each $n \in \mathbb{N}$, write $d(n) = p^{k(n)} m_n$ where m_n is not divisible by p. Suppose $f_n = g_1^{p^{e_1} m_1} \dots g_t^{p^{e_t} m_t}$ for irreducible polynomials $g_1, \dots, g_t$, with $m_1, \dots, m_t$ not divisible by p. Then $f_n^{d(n)} = g_1^{p^{e_1} p^{k(n)} m_1 m(n)} \dots g_t^{p^{e_t} p^{k(n)} m_t m(n)}$. Since $f_n^{d(n)}$ is computable by an arithmetic circuit of size at most $s(n)$ and has degree at most $s^2(n)$, Kaltofen's factoring algorithm of [Kal89] guarantees the existence of t arithmetic circuits $C_1, \dots, C_t$ over $\mathbb{F}$ of size $\mathsf{poly}(s(n), \log |\mathbb{F}|)$ computing the polynomials $g_1^{p^{e_1+k(n)}}, \dots, g_t^{p^{e_t+k(n)}}$. Define new circuits $C_i' = C_i^{m_i}$, for $1 \leq i \leq t$. Observe that the product $C_1' \dots C_t'$ of these circuits computes the polynomial $f_n^{p^{k(n)}}$, and has size $\mathsf{poly}(s(n))$. □

To simplify notation, we will often omit the subscripts $\mathbb{F}$ and $\mathbb{Z}$ for arithmetic circuit classes ASize, ASizeDeg, ASizeDegPow, and ASizeDegMul, when the underlying domain is clear from the context. For instance, in this work we will never have occasion to use ASizeDegMul over finite fields or ASizeDegPow over $\mathbb{Z}$.

6.2.2 Polynomials computable in NE: The class ml-NE

Fix an arbitrary finite field $\mathbb{F} = \mathbb{F}_{p^k}$ of characteristic p. Let $f = \{f_n(x_1, \ldots, x_n)\}_{n \geq 0}$ be an arbitrary family of polynomials over $\mathbb{F}$. For a polynomial f and a monomial m, denote by $\mathsf{coeff}(f, m)$ the coefficient of f at the monomial m (when f is written out as the sum of its monomials). For a family f of multilinear polynomials f_n over a finite field $\mathbb{F}$, define the following language:

$$\text{MONOMIAL}(f) = \{(a_1, \ldots, a_n, c) \mid a_1, \ldots, a_n \in \{0,1\},\ c \in \mathbb{F},\ \mathsf{coeff}(f_n, x_1^{a_1} \ldots x_n^{a_n}) = c\}.$$

We denote by $\text{MONOMIAL}(f_n)$ the restriction of $\text{MONOMIAL}(f)$ to the case of f_n, that is, $\text{MONOMIAL}(f_n)$ defines the coefficients of the n-variate polynomial f_n. For multilinear polynomial families f over the integers, the language $\text{MONOMIAL}(f)$ is defined similarly, with the natural change that the coefficient c be an integer in binary.

We say that a family of multilinear polynomials $f = \{f_n\}_{n \geq 0}$ over a finite field $\mathbb{F}$ or over integers $\mathbb{Z}$ is in the class ml-NE if $\text{MONOMIAL}(f) \in \mathsf{NE}$.

Observe that if $\text{MONOMIAL}(f) \in \mathsf{NE}$, then $\text{MONOMIAL}(f) \in \mathsf{coNE}$ also. Thus, we have $\text{MONOMIAL}(f) \in \mathsf{NE} \iff \text{MONOMIAL}(f) \in (\mathsf{NE} \cap \mathsf{coNE})$, and so $\mathsf{ml\text{-}NE} = \mathsf{ml\text{-}(NE \cap coNE)}$; the same equivalence holds also for the definition of ml-NE used by Jansen and Santhanam [JS12].

Remark 89. *The graph definition of* ml-NE *used by [JS12] and the monomial-coefficient definition of* ml-NE *given here are equivalent because of efficient interpolation: If we can compute each multilinear coefficient of f in* NE*, then we can compute all 2^n coefficients and use them to evaluate $f(x)$ on any given x; conversely, if we can compute the graph $(x, f(x))$ in* NE*, then*

we can compute $f(x)$ on all Boolean points $x \in \{0,1\}^n$ in NE*, and use polynomial interpolation to recover all multilinear coefficients. We use the coefficient definition in our work just for convenience, as it makes it obvious that one can evaluate polynomials over extensions of finite fields (which we will need in the context of using hard polynomials to construct pseudo-random generators).*

6.2.3 Computably subexponential and superpolynomial bounded classes

We call a function $\alpha\colon \mathbb{N} \to \mathbb{R}^{\geq 0}$ *computably super-constant*, denoted by $\alpha(n) \in \omega_c(1)$, if there exists computable, monotone non-decreasing function $\alpha'\colon \mathbb{N} \to \mathbb{R}^{\geq 0}$ with $\alpha'(n) \to \infty$ such that, for all sufficiently large $n \in \mathbb{N}$, $\alpha(n) \geq \alpha'(n)$. Without loss of generality, we may always assume that our computably super-constant functions $\alpha(n)$ are computable in time $\mathsf{poly}(n)$ (see Lemma 90 below).

The standard definition of NSUBEXP is $\bigcap_{\varepsilon>0} \mathsf{NTIME}[2^{n^\varepsilon}]$. This is problematic when we need to run a padding argument for some language in this class, because the amount of padding required will depend on ε. There is no uniform way to produce the specific ε required for a particular amount of padding, so any padding argument must rely on advice to get the correct value of ε. We use our notion of computably super-constant functions to trade a more restricted class for freedom from advice, and define computably subexponential-time classes as follows: We say that a language L is in nondeterministic computably subexponential time, denoted $L \in \mathsf{NSUBEXP}_c$, if $L \in \mathsf{NTIME}[2^{n^{1/\alpha(n)}}]$ for some $\alpha(n) \in \omega_c(1)$. We denote by $\mathsf{superpoly}_c$ any function $n^{\alpha(n)}$ for some $\alpha(n) \in \omega_c(1)$.

We will actually need these computably super-constant functions to be *efficiently* computable. However, we can always get an efficiently computable super-constant function, given a computable one. So, without loss of generality, we may always assume that our computably super-constant functions $\alpha(n)$ are computable in time $\mathsf{poly}(n)$. A proof appears below, but can be skipped on a first reading.

Lemma 90. *For every computably super-constant function $\alpha(n)$, there exists a related super-constant function $\beta(n)$ that is computable in time* $\mathsf{poly}(n)$.

Proof. Define $\beta(n)$ to be the maximum $\alpha(m)$ over all $m \leq n$ such that $\alpha(m)$ is computed in fewer than n steps. □

Later we shall need the following simple lemma.

Lemma 91. *For any $\alpha \in \omega_c(1)$, there exists a non-decreasing $\gamma \in \omega_c(1)$ such that $\gamma(n) \leq \alpha(n)$ and $\gamma(n^2) \leq \gamma(n)+1$. As a consequence, we have $\gamma(n^k) \leq \gamma(n) + \lceil \log k \rceil$.*

Proof. By definition, there exists a computable non-decreasing $\alpha'(n)$ such that for some n_0, $\forall n \geq n_0, \alpha(n) \geq \alpha'(n)$. Assume, wlog, that $n_0 \geq 1$. Define $\gamma(n) = \min\{\alpha'(n), \gamma(\lfloor\sqrt{n}\rfloor)+1\}$ for $n > n_0$; set $\gamma(n) = 0$ for $n \leq n_0$. Thus, $\gamma(n) \leq \alpha'(n) \leq \alpha(n)$ for $n > n_0$, and $\gamma(n) = 0 \leq \alpha(n)$ otherwise.

To see that the resulting γ is non-decreasing, argue by induction on n. Assume that $\gamma(k-1) \leq \gamma(k)$ for all $k < n$. As $\gamma(k) = 0$ for $k \leq n_0$, and thus non-decreasing up to n_0, let $n > n_0$. If $\gamma(n) = \alpha'(n)$, then since α' is non-decreasing, $\gamma(n-1) \leq \min\{\alpha'(n-1), \gamma(\lfloor\sqrt{n-1}\rfloor+1\} \leq \alpha'(n-1) \leq \alpha'(n) = \gamma(n)$. Otherwise if $\gamma(n) = \gamma(\lfloor\sqrt{n}\rfloor)+1 \leq \alpha'(n)$, then by induction hypothesis $\gamma(\lfloor\sqrt{n-1}\rfloor) \leq \gamma(\lfloor\sqrt{n}\rfloor)$, so $\gamma(n-1) \leq \min\{\alpha'(n-1), \gamma(\lfloor\sqrt{n-1}\rfloor)+1\} \leq \gamma(\lfloor\sqrt{n-1}\rfloor)+1 \leq \gamma(\lfloor\sqrt{n}\rfloor)+1 = \gamma(n)$.

To show that $\gamma(n) \to \infty$, note that either there are infinitely many n such that $\gamma(n) = \gamma(\sqrt{n})+1$ or for all large enough n we have $\gamma(n) = \alpha'(n)$. In either case, γ is unbounded.

Finally, by definition, $\gamma(n^2) \leq \min\{\alpha'(n^2), \gamma(n)+1\} \leq \gamma(n)+1$. To show $\gamma(n^k) \leq \gamma(n) + \lceil \log k \rceil$, for $k = 2^r$, $1 \leq r \in \mathbb{N}$, proceed by induction on r. For k which is not a power of 2, if $2^{r-1} < k < 2^r$, then $\gamma(n^k) \leq \gamma(n^{2^r}) \leq \gamma(n) + r = \gamma(n) + \lceil \log k \rceil$ by monotonicity of γ. □

6.2.4 Derandomization of Polynomial Identity Testing

Let $\mathbb{F}$ be any finite field. For $D \in \{\mathbb{Z}, \mathbb{F}\}$, the Polynomial Identity Testing problem over D, denoted PIT_D, is the task to decide if a given arithmetic circuit C over D computes the

identically zero polynomial (when the polynomial C is expanded as the sum of monomials). Stating this definition using sums-of-monomials is crucial, because over a finite field $\mathbb{F}$, testing if a polynomial *agrees with* the zero polynomial for all inputs over $\mathbb{F}$ is actually coNP-complete. The sum-of-monomials definition that we use puts $\mathsf{PIT}_{\mathbb{F}} \in \mathsf{BPP}$, and thus it is meaningful to ask for a derandomization of this problem. The $\mathsf{low\text{-}PIT}_D$ variant is restricted to test only circuits that have degree (or, in the case of $\mathbb{Z}$, formal degree) less than their size.

A possible approach to solve PIT is via hitting set generators. A *hitting set generator* for PIT_D is a function $\mathscr{H}$, that on input 1^n, outputs a collection of t tuples $a_1, \ldots, a_t \in D^n$ such that, for every arithmetic circuit $C(x_1, \ldots, x_n)$ over D of size at most n, if $C \not\equiv 0$, then there is at least one $i \in [t]$ where $C(a_i) \neq 0$. In case of a finite field $\mathbb{F}$, we allow the tuples $a_1, \ldots, a_t$ to come from $(\mathbb{F}')^n$ for some extension field $\mathbb{F}'$ of $\mathbb{F}$. Such a collection of t tuples is called a *hitting set* of size t for PIT instances of size n.

A hitting set for $\mathsf{low\text{-}PIT}_D$ instances of size n is defined in the natural way by restricting to the circuits of size n that additionally have (formal) degree at most n. We will need the following hardness to randomness result from pervious work, where [KI04] show that a hard polynomial $f \notin \mathsf{ASizeDegPow}_{\mathbb{F}}[\mathsf{superpoly}_c]$ can be used to derandomize $\mathsf{low\text{-}PIT}_{\mathbb{F}}$ in subexponential time (via a hitting set).

Theorem 92 (Hardness to Hitting Arithmetic Circuits [KI04], [JS12])**.** *Let $D \in \{\mathbb{Z}, \mathbb{F}\}$, where $\mathbb{F}$ is any finite field. There is a constant $c > 0$ and a deterministic algorithm that, given oracle access to an n-variate multilinear polynomial f_n over D and parameter $0 \leq s \leq 2^n$, outputs a collection H of $t := s^{n^c}$ tuples in $(D')^m$, for $m := s^{1/c}$, satisfying the following:*

- *case of $\mathbb{Z}$: $D' = \mathbb{Z}$, and whenever f_n is not in $\mathsf{ASizeDegMul}_{\mathbb{Z}}[s]$, then H is a hitting set for $\mathsf{low\text{-}PIT}_{\mathbb{Z}}$ instances of size m.*
- *case of $\mathbb{F}$: D' is an extension field of $\mathbb{F}$ of size m^c, and if f_n is not in $\mathsf{ASizeDegPow}_{\mathbb{F}}[s]$, then H is a hitting set for $\mathsf{low\text{-}PIT}_{\mathbb{F}}$ instances of size m.*

The running time of the algorithm is $\mathsf{poly}(t)$.

We shall omit the subscripts $\mathbb{F}$ and $\mathbb{Z}$ in PIT and low-PIT, when it is clear from the context.

6.2.5 Derandomization of promise-BPP

Derandomizing promise-BPP is equivalent to getting an efficient algorithm for estimating the acceptance probability of a given Boolean circuit $C(x_1,\ldots,x_n)$ of size n to within an additive error $1/n$; the latter problem is called Circuit Acceptance Probability Problem (CAPP). We use the notation promise-BPP $\subseteq$ NSUBEXP$_c$ to mean that there is a nondeterministic Turing machine that runs in computably subexponential time, and solves CAPP with an additive approximation error $1/n$. That is, on a given circuit $C(x_1,\ldots,x_n)$, the nondeterministic machine has at least one accepting computation, and every accepting computation yields a value r such that $|\mathbf{Pr}_{a\in\{0,1\}^n}[C(a)=1]-r|\leq 1/n$.

We say that promise-BPP $\subseteq$ ro$_\star$-NSUBEXP$_c$ if a NSUBEXP$_c$ algorithm correctly approximates CAPP only robustly often.

A possible approach to solve CAPP is via an efficient pseudorandom generator. A *pseudorandom generator (PRG)* is a function $\mathscr{G}$ that, on input 1^n, outputs a collection of t strings $a_1,\ldots,a_t\in\{0,1\}^n$ such that $\left|\mathbf{Pr}_{z\in\{0,1\}^n}[C(z)=1]-\mathbf{Pr}_{i\in[t]}[C(a_i)=1]\right|\leq 1/n$, for every Boolean circuit C of size n on n inputs. Such a collection of strings $a_1,\ldots,a_t$ is called a *discrepancy set* of size t for circuits of size n.

In the Boolean setting, it was shown by [BFNW93] that the truth table of a superpolynomial circuit-complexity Boolean function can be used to get a subexponential-size discrepancy set (in subexponential time).

Theorem 93 ([BFNW93])**.** *There exist a constant $c>0$ and deterministic algorithm that, given a 2^n-bit string T and a parameter $0\leq s\leq 2^n/n$ outputs a collection of $t:=2^{n^c}$ binary strings of length $m:=s^{1/c}$ satisfying the following: If T (when viewed as the truth table of a Boolean n-variate function) requires circuit size s, then the collection of these t strings is a discrepancy set for circuits of size m. The running time of this algorithm is* poly(t).

6.3 Robustness

To establish equivalences between circuit lower bounds and derandomization, we need notions that are intermediate between infinitely-often and almost-everywhere. In the spirit of [FS11], we define "robust" notions of containment and separation for complexity classes. Our robust ranges are some fixed *computably super-polynomial* function in length, whereas the ranges of [FS11] are for every fixed polynomial function. The reason we require this alternative notion is because the simulation steps of our arguments use superpolynomial amounts of padding. The [FS11] definition handles fixed polynomial-time translations or translations with fixed polynomial advice, but not superpolynomial translations. Our definition is an attempt to make the minimal extention possible to [FS11] that can handle superpolynomial translations. Thus we do not expect our results to hold under the [FS11] robustness notions, without major technical innovation in hardness-randomness tradeoffs that dispenses with the necessity for superpolynomial padding.

For functions $l, r : \mathbb{N} \to \mathbb{N}$, called left and right "interval functions", define the (l,r)-*core* of a set $S \subseteq \mathbb{N}$ to be the set of intervals $[l(m), r(m)]$ that are entirely contained in S, i.e., $\mathsf{core}(S) = \{m \in \mathbb{N} \mid \forall n \in \mathbb{N}\ (l(m) \leq n \leq r(m) \implies n \in S)\}$. A set S is called (l,r)-*robust* if the (l,r)-core of S is infinite. Finally, we say that S is *computably robust* if S is $(m^{1/\alpha(m)}, m^{\alpha(m)})$-robust for some $\alpha \in \omega_c(1)$. For brevity, we shall call such a set S simply α-*robust*.

6.3.1 Robust inclusions.

For a language L and a complexity class $\mathscr{C}$, we say that L is *uniform robustly often in* $\mathscr{C}$, denoted $L \in \mathsf{ro}_\star\text{-}\mathscr{C}$, if there is a language $N \in \mathscr{C}$ such that the set $S = \{n \in \mathbb{N} \mid L_n = N_n\}$ is computably robust, where $L_n = L \cap \{0,1\}^n$ is the nth slice of L. Our definition is "uniform" compared to the notion of [FS11] because the notion defined there has interval lengths that are defined by every fixed polynomial function — this is an infinite (but very regular) set of functions giving interval lengths. Our robust sets have a fixed pair of interval functions.

We say that a family $f = \{f_n\}$ of multilinear polynomials (over a finite field $\mathbb{F}$ or over integers $\mathbb{Z}$) is in $\mathsf{ro}_\star$-ml-NE, *robustly often* ml-NE, if, for some NE machine M, the set $S = \{n \in \mathbb{N} \mid M \text{ correctly decides } \textsc{monomial}(f_n)\}$ is computably robust.

6.3.2 Robust promise classes.

For a language L and a *semantic* complexity $\mathscr{C}$, we say that L is in *uniform robustly promise* $\mathscr{C}$, denoted $L \in \mathsf{rp}_\star$-$\mathscr{C}$, if there is a Turing machine M such that

$$S = \{n \in \mathbb{N} \mid \text{ for all } x \in \{0,1\}^n, M(x) \text{ is a } \mathscr{C}\text{-type machine and } M(x) \text{ decides if } x \in L_n\}$$

is computably robust.

In general, $\mathsf{ro}_\star$-$\mathscr{C}$ and $\mathsf{rp}_\star$-$\mathscr{C}$ are different for a semantic class $\mathscr{C}$. For example, $L \in \mathsf{ro}_\star$-(NE $\cap$ coNE) if there is a language $N \in$ (NE $\cap$ coNE) that robustly often agrees with L. That is, there is a pair of NE machines M_1 and M_2 such that, for every $x \in \{0,1\}^n$, exactly one of M_1 and M_2 accepts x, and $S = \{n \in \mathbb{N} \mid M_1 \text{ decides } L_n \text{ and } M_2 \text{ decides } \overline{L_n}\}$ is computably robust, where $\overline{L_n}$ is the complement of L_n.

In contrast, $L \in \mathsf{rp}_\star$-(NE $\cap$ coNE) if there is a pair of NE machines M_1 and M_2 such that $S = \{n \in \mathbb{N} \mid M_1 \text{ decides } L_n \text{ and } M_2 \text{ decides } \overline{L_n}\}$ is computably robust. Note that, in the second case, the machines M_1 and M_2 may not be "complementary" on the input lengths outside of S, and so we may not have any language $N \in$ (NE $\cap$ coNE) that agrees with L robustly often: in the $\mathsf{rp}_\star$- case, the promise is not required to hold for slices outside the robust set.

6.3.3 Significant separations.

To complement the inclusion types above, we define *uniform significant separations* denoted $\mathsf{ro}_\star$-$\mathscr{C} \not\subseteq$ SIZE$[s(n)]$. We write $\mathsf{ro}_\star$-$\mathscr{C} \not\subseteq$ SIZE$[s(n)]$ if there is a language $L \in \mathsf{ro}_\star$-$\mathscr{C}$ over computably robust set S such that L_n cannot be computed by a circuit of size $s(n)$ for infinitely many values $n \in \mathsf{core}(S)$.

Intuitively, a uniform significant separation means that we can always locate hard lengths for the separated class in the "middle" of large, computably robust intervals. This is different from the [FS11] notion, which says (intuitively) that hard lengths are never too far apart. Under our definition, if the robust set comes with a promise, this means that hard lengths are located in the center of large ranges where the promise holds. This is what our arguments for equivalence will hinge on. The generalization of this definition to arithmetic circuits and uniform robust promise classes is obvious. For example, we define two uniform significant separations below.

We say that $\mathsf{ro}_\star\text{-ml-NE} \not\subseteq \mathsf{ASize}[s(n)]$ if there is a polynomial family $f = \{f_n\}$ where for each n we have $f_n \in \mathsf{ro}_\star\text{-ml-NE}$, with $\textsc{monomial}(f)$ correctly decided by some NE machine on a computably robust set S, such that f_n cannot be computed by an arithmetic circuit of size $s(n)$ for infinitely many values $n \in \mathsf{core}(S)$. The case of other arithmetic circuit classes (ASizeDeg and ASizeDegPow) is similar.

We say that $\mathsf{rp}_\star\text{-}(\mathsf{NE} \cap \mathsf{coNE}) \not\subseteq \mathsf{SIZE}[s(n)]$ if there is a promise problem $L \in \mathsf{rp}_\star\text{-}(\mathsf{NE} \cap \mathsf{coNE})$, with a pair of NE machines correctly deciding L and $\bar{L}$ over a computably robust set S of input lengths, such that L_n cannot be computed by a Boolean circuit of size $s(n)$ for infinitely many input lengths $n \in \mathsf{core}(S)$.

6.3.4 Closure properties of robust sets

This subsection establishes several useful closure properties of computably robust sets. This subsection should be skipped on a first reading and returned to once the need for these closure properties has been demonstrated by our applications. The first-time reader should continue with Section 6.4 on page 164, which discusses a useful PSPACE-complete function.

Definition 94 (Polynomial-time Honest Turing Reductions)**.** Let L and L' be langauges. We say that L' reduces to L by a polynomial-time honest Turing reduction if L' can be decided by a Turing machine M equipped with an oracle for L, where:

- M runs in polynomial time

- For any input length n, M does not call the oracle with queries of length less than n

Computable Robustness is closed under honest polynomial time Turing reductions. Formally:

Lemma 95 (Analog to Lemma 1 of [FS11])**.** *Let L be a paddable langauge. Let L' be a language that reduces to L by a polynomial-time honest Turing reduction, and let C be a complexity class closed under polynomial-time Turing reductions. Then,*

$$L \in \mathsf{ro}_\star\text{-}C \implies L' \in \mathsf{ro}_\star\text{-}C$$

Proof. If $L \in \mathsf{ro}_\star$-C, then there exists some computably robust S such that $L \in C$ on S. Since C is closed under polynomial-time Turing reductions, then by definition the machine attempting to decide L' by running the L oracle as a "subroutine", $M_{L'}$, is in C. But since we only have $L \in \mathsf{ro}_\star$-C, we must ensure that the input lengths of the L-oracle subroutine are in S, robustly-often, to show that $M_{L'}$ actually decides L'.

First, by the polynomial time bound on the reduction, there is some fixed integer k such that the length of queries to L does not exceed $O(n^k)$. By the honesty of the reduction, the length of queries to L never drops below n. Since L is paddable, we can modify $M_{L'}$ to fix the size of queries to L to n^k.

Since S is computably robust, we have that there are infinitely many m such that, $\forall m'$ such that $m^{1/\alpha(m)} \leq m' \leq m^{\alpha(n)}$, $m' \in S$. Then, given that on input length n every query to L will be of length n^k, we have that $M_{L'}$ decides L' on inputs of length n exactly when:

$$m^{1/\alpha(m)} \leq n^k \leq m^{\alpha(m)}$$

For some $m \in \mathsf{core}(S)$. Define $\beta := \alpha(m)/k$. The $(n^{1/\beta(n)}, n^{\beta(n)})$-robust set $S' \subset S$ produced by taking β intervals around each core of S is a subset of the above intervals. Therefore, $M_{L'}$ decides L' on S', proving that $L' \in \mathsf{ro}_\star$-C.

□

Lemma 96 (Closure under some super-polynomial translations)**.** *Let S be α-robust, for some $\alpha \in \omega_c(1)$ such that, for every $k \geq 1$, $\alpha(n^k) \leq \alpha(n) + \lceil \log k \rceil$. Let $t(n) = n^{\beta(n)}$ for $\beta(n) := \sqrt{\alpha(n)}/2$. Then $t^{-1}(S)$ is β-robust.*

Proof. Let $\bar{n}$ be an arbitrary element of the $(n^{1/\alpha(n)}, n^{\alpha(n)})$-core of S. Consider an interval of n's such that $\bar{n}^{1/\beta(\bar{n})} \leq n \leq \bar{n}^{\beta(\bar{n})}$. We will show that $\bar{n}^{1/\alpha(\bar{n})} \leq t(n) \leq \bar{n}^{\alpha(\bar{n})}$, implying that $t^{-1}(S)$ is β-robust.

First we obtain the upper bound on $t(n)$, by substituting the maximum value of n into the definition of $t(n)$:

$$t(n) \leq \left(\bar{n}^{\beta(\bar{n})}\right)^{\beta\left(\bar{n}^{\beta(\bar{n})}\right)}$$

Examining the exponent, note that:

$$\begin{aligned}
\beta(m^{\beta(m)}) &= \sqrt{\alpha(m^{\beta(m)})/4} \\
&\leq \sqrt{(\alpha(m) + \lceil \log \beta(m) \rceil)/4} && \text{properties of } \alpha \\
&\leq \sqrt{(\alpha(m) + \lceil \log \alpha(m) \rceil)/4} \\
&\leq \sqrt{\alpha(m)/2}
\end{aligned}$$

Returning to our bound on $t(n)$ and substituting:

$$\begin{aligned}
t(n) &\leq \left(\bar{n}^{\beta(\bar{n})}\right)^{\sqrt{\alpha(\bar{n})/2}} \\
&\leq \left(\bar{n}^{\sqrt{\alpha(\bar{n})/4}}\right)^{\sqrt{\alpha(\bar{n})/2}} \\
&\leq \bar{n}^{\alpha(\bar{n})}
\end{aligned}$$

Similarly, we check the lower bound on $t(n)$ by substituting the minimum value of n into

the definition of $t(n)$:

$$\begin{aligned} t(n) &= n^{\beta(n)} \\ &\geq \left(\bar{n}^{1/\beta(\bar{n})}\right)^{\beta\left(\bar{n}^{1/\beta(\bar{n})}\right)} \\ &\geq \bar{n}^{\left(\beta\left(\bar{n}^{1/\beta(\bar{n})}\right)/\beta(\bar{n})\right)} \end{aligned}$$

Looking at the exponent and expanding the definition of β, we have:

$$\frac{\beta\left(\bar{n}^{1/\beta(\bar{n})}\right)}{\beta(\bar{n})} = \frac{\sqrt{\alpha\left(\bar{n}^{1/\beta(\bar{n})}\right)/4}}{\sqrt{\alpha(\bar{n})/4}} \geq \frac{1}{\alpha(\bar{n})}$$

The last inequality holds because $\sqrt{\alpha(\bar{n}^{1/\beta(\bar{n})})/4} > 1$, and $\sqrt{\alpha(\bar{n})/4} < \alpha(\bar{n})$. Then $t(n) \geq \bar{n}^{1/\alpha(\bar{n})}$ follows, as required. □

Lemma 97 (Closure under intervals around cores)**.** *Let S be γ-robust, for some $\gamma \in \omega_c(1)$ such that, for every $k \geq 1$, $\gamma(n^k) \leq \gamma(n) + \lceil \log k \rceil$. Let $H \subseteq \mathsf{core}(S)$ be an infinite set, and let $\alpha \in \omega_c(1)$ be the hardness parameter for lengths in H, again such that for every $k \geq 1$, $\alpha(n^k) \leq \alpha(n) + \lceil \log k \rceil$.*

Then there exist "low" and "high" functions T_l and T_h such that, for

$$I(n) := [T_l(n), T_h(n)] \qquad \textit{and} \qquad G := \{n \mid I(n) \subseteq S \textit{ and } I(n) \cap H \neq \emptyset\}$$

we have the following:

1. *G is computably robust.*
2. *$T_l(n)^{\alpha(T_l(n))} \in n^{\omega_c(1)}$ (hardness is maintained at low lengths).*
3. *$T_h(n) \in n^{o_c(1)}$ (circuits don't get too large at high lengths).*

Proof. Set $\beta'(n) = \min\{\sqrt{\alpha(n)}, \gamma(n)\}$, and let $\beta(n)$ be the "slowed down" version of $\beta'(n)$, using Lemma 91.

Define $T_l(n) := n^{2/\beta(n)}$ and $T_h(n) := n^{1/\sqrt{\beta(n)}}$.

Claim 98. *The functions T_l and T_h defined above satisfy assertions (2) and (3) in the statement of the lemma.*

Proof. Assertion (3) is obviously satisfied by definition. For assertion 2, we examine $T_l(n)^{\alpha(T_l(n))}$:

$$T_l(n)^{\alpha(T_l(n))} = \left(n^{2/\beta(n)}\right)^{\alpha(n^{2/\beta(n)})} = n^{2\alpha(n^{2/\beta(n)})/\beta(n)}.$$

Looking at the exponent, we have:

$$\frac{2\alpha(n^{2/\beta(n)})}{\beta(n)} \geq \frac{2(\alpha(n) - \log(\beta(n)))}{\beta(n)} \geq \frac{\alpha(n)}{\beta(n)} \geq \sqrt{\alpha(n)},$$

where the first inequality is by the assumption on α, and the last inequality is by the definition of β. Thus, $T_l(n)^{\alpha(T_l(n))} \in n^{\omega_c(1)}$, as required. □

Fix an arbitrary $m \in H$, and consider the interval $J[m] := [m^{\sqrt{2\beta(m)}}, m^{\beta(m)/2}]$. Clearly, $J[m] \subseteq S$. We will show that every $n \in J[m]$ is in the set G (from the statement of the lemma).

First, we prove the following auxiliary claim.

Claim 99. *For every $n \in J[m]$, we have $\beta(m) \leq \beta(n) \leq 2\beta(m)$.*

Proof. Upper-bounding $\beta(n)$, we get:

$$\beta(n_{\max}) = \beta(m^{2\beta(m)/2}) \leq \beta(m) + \lceil \log(\beta(m)/2) \rceil \leq 2\beta(m)$$

. Lower bounding $\beta(n)$, we get $\beta(n_{\min}) = \beta(m^{\sqrt{2\beta(m)}}) \geq \beta(m)$. □

Now we can show

Claim 100. *For every $n \in J[m]$, we have $I(n) \subseteq S$ and $m \in I(n)$. So, every $n \in J[m]$ is in G.*

Proof. This amounts to verifying the following chain of inequalities:

$$m^{1/\beta(m)} \leq T_l(n) \leq m \leq T_h(n) \leq m^{\beta(m)},$$

which we do below, one inequality at a time.

1. $m^{1/\beta(m)} \leq T_l(n)$: We have

$$\begin{aligned}
T_l(n) &= n^{2/\beta(n)} \\
&> n^{2/2\beta(m)} && \text{Claim 99} \\
&> \left(m^{\sqrt{2\beta(m)}}\right)^{2/2\beta(m)} && \text{minimize base} \\
&> m^{\sqrt{2\beta(m)}/\beta(m)} && \text{simplify} \\
&> m^{1/\beta(m)} && \beta(m) \in \omega_c(1)
\end{aligned}$$

2. $T_l(n) \leq m$: We have

$$\begin{aligned}
T_l(n) &= n^{2/\beta(n)} \\
&< n^{2/\beta(m)} && \text{Claim 99} \\
&< \left(m^{\beta(m)/2}\right)^{2/\beta(m)} && \text{maximize base} \\
&< m && \text{simplify}
\end{aligned}$$

3. $m \leq T_h(n)$: We have

$$\begin{aligned} T_h(n) &= n^{1/\sqrt{\beta(n)}} \\ &> n^{1/\sqrt{2\beta(m)}} && \text{Claim 99} \\ &> \left(m^{\sqrt{2\beta(m)}}\right)^{1/\sqrt{2\beta(m)}} && \text{minimize base} \\ &> m && \text{simplify} \end{aligned}$$

4. $T_h(n) \leq m^{\beta(m)}$: We have

$$\begin{aligned} T_h(n) &= n^{1/\sqrt{\beta(n)}} \\ &< n^{1/\sqrt{\beta(m)}} && \text{Claim 99} \\ &< \left(m^{\beta(m)/2}\right)^{1/\sqrt{\beta(m)}} && \text{maximize base} \\ &< m^{\beta(m)/2\sqrt{\beta(m)}} && \text{simplify} \\ &< m^{\beta(m)} \end{aligned}$$

□

To argue that G is computably robust, we will construct an appropriate interval function δ depending on β, and show how to choose a core element m_0 within each interval $J[m]$ so that

$$m^{\sqrt{2\beta(m)}} \leq m_0^{1/\delta(m_0)} \leq m \leq m_0^{\delta(m_0)} \leq m^{\beta(m)/2}. \tag{6.1}$$

The chain of inequalities in Eq. (6.1) means that the δ-interval around m_0 is contained within $J[m]$, and hence, by Claim 100, is contained within G. Since there are infinitely many $m \in H$, we get that G is δ-robust.

First we set m_0. Take the geometric mean of the exponents of m on the endpoints of the range above: $(\sqrt{2\beta(m)})^{1/2} \cdot (\beta(m)/2))^{1/2} = (2\beta(m))^{1/4} \cdot \beta(m)^{1/2} \cdot 2^{-1/2} = 2^{-1/4} \cdot \beta(m)^{3/4}$,

and set $m_0 = m^{2^{-1/4}\cdot\beta(m)^{3/4}}$; this will ensure that m_0 is centrally located.

To find a setting of $\delta(n) \in \omega_c(1)$, consider functions of the form: $\delta(n) \sim x(z\beta(n))^{1/k}$. Given that $\beta(n) \in \omega_c(1)$ and for x,k,z constants, we have that $\delta \in \omega_c(1)$. We will obtain an appropriate triple of constants such that Eq. (6.1) holds.

First consider the low end, $m^{\sqrt{2\beta(m)}} < m_0^{1/\delta(m_0)}$. Substitute and simplify to bound $m_0^{1/\delta(m_0)}$ below:

$$\begin{aligned} m_0^{1/\delta(m_0)} &= \left(m^{\beta(m)^{3/4}/2^{1/4}}\right)^{1/x(z\beta(m_0))^{1/k}} \\ &= m^{\beta(m)^{3/4}/2^{1/4}x(z\beta(m_0))^{1/k}} \end{aligned}$$

Examining the exponent and substituting for m_0, we have:

$$\begin{aligned} \frac{\beta(m)^{3/4}}{2^{1/4}x(z\beta(m^{2^{-1/4}\cdot\beta(m)^{3/4}}))^{1/k}} &> \frac{\beta(m)^{3/4}}{2^{1/4}x(z(\beta(m)+\log(2^{-1/4}\cdot\beta(m)^{3/4})))^{1/k}} \\ &> \frac{\beta(m)^{3/4}}{2^{1/4}x(z2\beta(m))^{1/k}} \end{aligned}$$

This last expression is tractable. It implies that we require x,z,k such that:

$$\frac{\beta(m)^{3/4}}{2^{1/4}x(z2\beta(m))^{1/k}} > (2\beta(m))^{1/2}$$

We derive an analogous constraint for the high end inequality, $m_0^{\delta(m_0)} < m^{\beta(m)/2}$.

$$\begin{aligned} m_0^{\delta(m_0)} &= \left(m^{\beta(m)^{3/4}/2^{1/4}}\right)^{x(z\beta(m_0))^{1/k}} \\ &= m^{(\beta(m)^{3/4}x(z\beta(m_0))^{1/k})/(2^{1/4})} \end{aligned}$$

Examining the exponent and substituting for m_0:

$$\frac{\beta(m)^{3/4}x(z\beta(m^{2^{-1/4}\cdot\beta(m)^{3/4}}))^{1/k}}{2^{1/4}} < \frac{\beta(m)^{3/4}x(z(\beta(m)+\log(2^{-1/4}\cdot\beta(m)^{3/4})))^{1/k}}{2^{1/4}} < \frac{\beta(m)^{3/4}x(z2\beta(m))^{1/k}}{2^{1/4}}$$

This yields the constraint:

$$\frac{\beta(m)^{3/4}x(z2\beta(m))^{1/k}}{2^{1/4}} < \frac{\beta(m)}{2}$$

Now we can begin to set the constants. First, we make the obvious choice for z, which is $\frac{1}{2}$, because this allows us to combine all terms involving $\beta(m)$ on both the high and the low end of the ranges, resulting in the following simplifications:

$$\text{[low end]:}\quad \frac{\beta(m)^{3/4-1/k}}{x2^{1/4}} > (2\beta(m))^{1/2} \qquad \text{and} \qquad \text{[high end]:}\quad \frac{x\beta(m)^{3/4+1/k}}{2^{1/4}} < \frac{\beta(m)}{2}.$$

Setting k is now straightforward. We want the powers of $\beta(m)$ appearing in these exponents to be appropriately bounded, so we require that $3/4-1/k > 1/2$ and $3/4+1/k < 1$. Any $k > 4$ will work, so set $k = 5$.

Finally we need to set x to take care of constant multiples of $\beta(m)$ occurring in the exponents. On the low end, we have less slack and we want: $x \cdot 2^{1/4} \leq 1/2$. So set $x = 2^{-5/4}$. Substituting into the low end, we have:

$$\frac{\beta(m)^{3/4-1/5}}{2^{-5/4}\cdot 2^{1/4}} = \frac{\beta(m)^{3/4-1/5}}{2^{-1}} = 2(\beta(m)^{3/4-1/5}) > (2\beta(m))^{1/2},$$

as required. Now we check the high end:

$$\frac{x\beta(m)^{3/4+1/5}}{2^{1/4}} = \frac{2^{-5/4}\cdot\beta(m)^{3/4+1/5}}{2^{1/4}} = \frac{\beta(m)^{3/4+1/5}}{2^{6/4}} < \frac{\beta(m)}{2}.$$

Therefore, setting $m_0 = m^{2^{-1/4}\cdot\beta(m)^{3/4}}$ and $\delta(n) = \frac{((1/2)\beta(n))^{1/5}}{2^{5/4}}$ suffices to show that G is

computably robust. □

6.4 PSPACE-complete polynomial

To prove circuit lower bounds from the assumption that $\mathsf{PIT}_{\mathbb{Z}}$ is easy, [KI04] used two facts: (1) the permanent function over $\mathbb{Z}$ is #P-complete [Val79] and hence, by Toda's theorem [Tod91], also PH-hard, and (2) a $\mathsf{PIT}_{\mathbb{Z}}$ algorithm allows one to test if a given arithmetic circuit computes the permanent. We can't use the same approach over finite fields because no analogue of Toda's theorem is known there, i.e., it is open whether $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{Mod}_k\mathsf{P}}$, for some (prime) $k \in \mathbb{N}$.

Instead, we will use a multilinear polynomial family $f = \{f_n\}$, obtained by arithmetizing the PSPACE-complete language TQBF, which will be PSPACE-complete over every finite field $\mathbb{F}$. Using the $\mathsf{PSPACE} = \mathsf{IP}$ proof ideas [LFKN92, Sha92], we show how a $\mathsf{PIT}_{\mathbb{F}}$ algorithm allows one to test whether a given arithmetic circuit computes some power f_n^d (for small $d \in \mathbb{N}$) of this PSPACE-complete polynomial over $\mathbb{F}$.

The first step, arithmetizing TQBF to get a multilinear polynomial f_n that is PSPACE-complete, already appears in work of Trevisan and Vadhan [TV07] to simplify and optimize the proofs in [IW01] (in particular, by removing the need for Toda's theorem in the arguments of [IW01]). For the second step, using $\mathsf{PIT}_{\mathbb{F}}$ algorithm to test if a given arithmetic circuit computes some power f_n^d, we generalize a standard IP protocol for TQBF so that we can handle prime powers $f_n^{p^k}$ of f_n, where prime p is the characteristic of the finite field $\mathbb{F}$. The latter turns out to be sufficient for our purposes due to Lemma 88, which allows us to assume, without loss of generality, that a small arithmetic circuit computing f_n^d in fact computes some prime power $f_n^{p^k}$ for $p^k \leq d$.

We provide more details on both of these steps in the following subsections.

6.4.1 Arithmetizing TQBF

First we review a construction of [TV07] (based on Shen's proof [She92] of the result PSPACE = IP of [LFKN92, Sha92]). Deciding the truth of a quantified Boolean formula in the prenex normal form (where the propositional part of the formula is a 3-CNF) is known to be PSPACE-complete. We first define the following "universal" CNF formula θ that can be instantiated to be any given 3-CNF formula ϕ in variables $\vec{x}$, thanks to the extra variables $\vec{a}$ that encode which 3-clauses are present in ϕ:

$$\theta_n(\vec{a},\vec{x}) := \bigwedge_{i,j,k\in[n],b_1,b_2,b_3\in\{0,1\}} \left(x_i^{b_1} \vee x_j^{b_2} \vee x_k^{b_3} \vee \neg a_{i,j,k,b_1,b_2,b_3}\right),$$

where the variable a_{i,j,k,b_1,b_2,b_3} is true iff the clause $x_i^{b_1} \vee x_j^{b_2} \vee x_k^{b_3}$ is present in ϕ; here we use x^0 and x^1 to denote $\bar{x}$ and x, respectively. Note that we need $O(n^3)$ variables $\vec{a}$ in order to specify any given 3-CNF in n variables. The universal QBF will be of the form: $\Phi_n(\vec{a}) := \exists x_1 \forall x_2 \ldots \exists/\forall x_n \;\; \theta_n(\vec{a},\vec{x})$.

Next we arithmetize Φ_n, getting a polynomial that agrees with Φ_n over all Boolean inputs (using 1 for true, and 0 for false). We begin by arithmetizing θ_n:

$$\hat{\theta}_n(\vec{a},\vec{x}) := \prod_{i,j,k\in[n],b_1,b_2,b_3\in\{0,1\}} \left(1-(1-\hat{x}_i^{b_1})(1-\hat{x}_j^{b_2})(1-\hat{x}_k^{b_3})\cdot a_{i,j,k,b_1,b_2,b_3}\right),$$

where $\hat{x}^0 = 1-x$ and $\hat{x}^1 = x$. It is easy to see that $\hat{\theta}_n$ agrees with θ_n on all Boolean values.

Next we arithmetize the quantifier sequence over $\vec{x}$. We define the following operators on polynomials, which apply either an appropriate quantifier or a linearization step:

$$\forall_{\mathrm{X}_i} q(\vec{a},\vec{x}) = q\restriction_{x_i\leftarrow 1} \cdot q\restriction_{x_i\leftarrow 0} \qquad \text{universal}$$

$$\exists_{\mathrm{X}_i} q(\vec{a},\vec{x}) = 1-(1-q\restriction_{x_i\leftarrow 1})\cdot(1-q\restriction_{x_i\leftarrow 0}) \qquad \text{existential}$$

$$\Lambda_{\mathrm{X}_i} q(\vec{a},\vec{x}) = x_i \cdot q\restriction_{x_i\leftarrow 1} + (1-x_i)\cdot q\restriction_{x_i\leftarrow 0} \qquad \text{linearization in } x_i$$

$$\Lambda_{\mathrm{A}_i} q(\vec{a},\vec{x}) = a_i \cdot q\restriction_{a_i\leftarrow 1} + (1-a_i)\cdot q\restriction_{a_i\leftarrow 0} \qquad \text{linearization in } a_i$$

Starting with $\exists x_1 \forall x_2 \ldots \exists/\forall x_n \;\; \hat{\theta}_n(\vec{a},\vec{x})$, we define a new formula by inserting the "linearization quantifiers" Λ as follows: between every pair of Qx_i and $Q'x_{i+1}$ quantifiers, where $Q \in \{\exists,\forall\}$ and $Q' = \neg Q$, we add a sequence of Λ_{A_j} for all variables a_j, and a sequence of Λ_{X_j} over all $1 \le j \le i$.

Next we think of all quantifiers in the new formula as operators on polynomials (as defined above), and apply these operators to the polynomial $\hat{\theta}_n(\vec{a},\vec{x})$ starting from the right-most operator. After each application of an operator, we get a new polynomial.

The iterated application of these operators results in a sequence of polynomials. Let $f_{m(n)} = \hat{\theta}_n$ be the first polynomial in this sequence, to which no operators have yet been applied. We let $m(n)$ be the number of operations required to arithmetize all quantifiers over n variables, and to linearize all relevant variables. It is clear that $m(n)$ is polynomial. At the end of this process, we will have a multilinear polynomial f_0 in variables $\vec{a}$ that equals the truth value (0 or 1) of any TQBF specified by a Boolean assignment to the $\vec{a}$ variables.

Finally, we define the following combination of the polynomials $f_{m(n)},\ldots,f_0$:

$$\widetilde{\mathrm{TQBF}}_n(\vec{z},\vec{a}) := \sum_{i=0}^{m(n)} z_i \cdot f_i(\vec{a},\vec{x}),$$

using new variables $\vec{z}$. Observe that by fixing exactly one $z_i = 1$ and the other $z_j = 0$, $j \neq i$, we can recover from $\widetilde{\mathrm{TQBF}}_n$ every polynomial f_i.

6.4.2 PSPACE-hardness of computing $\widetilde{\mathrm{TQBF}}_n$

The described construction of polynomials f_i and the polynomial $\widetilde{\mathrm{TQBF}}_n$ can be performed either over $\mathbb{Z}$ or in any given finite field $\mathbb{F}$. When carrying out the construction over $\mathbb{F}$, we still get that each polynomial f_i is $\{0,1\}$-valued (for $0,1 \in \mathbb{F}$), and that $f_0(\vec{a})$ computes the truth value of any given QBF (specified by the Boolean assignment to $\vec{a}$). Thus, if we can get our hands on the polynomials $\widetilde{\mathrm{TQBF}}_n$ (say by given arithmetic circuits computing $\widetilde{\mathrm{TQBF}}_n$), we will be able to solve TQBF.

Over a finite field $\mathbb{F}$ of characteristic p, to solve PSPACE-complete problems, it suffices to compute $\widetilde{\mathrm{TQBF}}_n^d$, for some $d = p^k$.

Lemma 101. *Let $\mathbb{F}$ be any finite field of characteristic p, and let $d = p^k$ for some $k \geq 0$. Then computing $\widetilde{\mathrm{TQBF}}_n^d$ over $\mathbb{F}$ is PSPACE-hard.*

Proof. Observe that $\widetilde{\mathrm{TQBF}}_n^d = (\sum_i z_i \cdot f_i)^d = \sum_i z_i^d \cdot f_i^d$ over $\mathbb{F}$. So we can use $\widetilde{\mathrm{TQBF}}_n^d$ to compute f_0^d. But since f_0 is Boolean-valued, the latter is as good as computing f_0 itself. □

We also observe that the language $\mathrm{MONOMIAL}(\widetilde{\mathrm{TQBF}}_n)$ is in *LINSPACE*.

Lemma 102. $\mathrm{MONOMIAL}(\widetilde{\mathrm{TQBF}}_n) \in$ *LINSPACE*.

Proof. For a fixed n, the polynomial $\widetilde{\mathrm{TQBF}}_n$ is a combination of multilinear polynomials $f_0, \ldots, f_{m(n)}$, for some $m(n) \in \mathsf{poly}(n)$, where each f_i is a polynomial in $\vec{a}$ (describing some QBF instance) and in $x_1, \ldots, x_j$, for some $j \leq n$. For each Boolean assignment to the variables $\vec{a}, \vec{x}$, we can compute the Boolean value $f_i(\vec{a}, \vec{x})$ in *LINSPACE*, since it is just the truth value of some TQBF instance specified by $\vec{a}, \vec{x}$. Let us denote the resulting Boolean function by f_i^{Bool}. The standard multilinear extension of f_i^{Bool} equals the polynomial f_i.

On the other hand, it is not hard to see that if a Boolean function g is in *LINSPACE*, then, for the multilinear extension g' of g, we have $\mathrm{MONOMIAL}(g') \in$ *LINSPACE*.

Finally, once we can compute the monomial language $\mathrm{MONOMIAL}(f_i)$ for all $0 \leq i \leq m(n)$, we can also easily compute $\mathrm{MONOMIAL}(\widetilde{\mathrm{TQBF}}_n)$. □

In the next subsection, we show that using a PIT algorithm, one can test if a given arithmetic circuit over a finite field $\mathbb{F}$ of characteristic p computes $\widetilde{\mathrm{TQBF}}_n^d$ for some $d = p^k$.

6.4.3 Testing arithmetic circuits for equality with $\widetilde{\mathrm{TQBF}}_n^d$

Suppose we are given a PIT algorithm. To test if a given arithmetic circuit C computes the polynomial $\widetilde{\mathrm{TQBF}}_n$, we first let C_i be obtained from C by fixing $z_i = 1$ and all $z_j = 0$ for $j \neq i$. These C_i's are candidate polynomials f_i's. We inductively test that each $C_i \equiv f_i$, starting at $i = m(n)$ and moving towards $i = 0$. For $i = m(n)$, we can actually construct an arithmetic formula $\hat{\theta}_n \equiv f_{m(n)}$ ourselves and use the PIT algorithm to test that $C_{m(n)} \equiv \hat{\theta}_n$. Then for each $i = m(n)-1,\ldots,0$, we check if $C_i \equiv \mathscr{O}_i[C_{i+1}]$, where $\mathscr{O}_i \in \{\Lambda_{\mathrm{X}_j}, \Lambda_{\mathrm{A}_j}, \exists_{\mathrm{X}_j}, \forall_{\mathrm{X}_j}\}$ is the operator used to construct f_i from f_{i+1}. Finally, we use the PIT algorithm to test if $C \equiv \sum_{i=1}^{m(n)} z_i \cdot C_i$. If each test above succeeds, then we get that $C_i \equiv f_i$ for all i, and that $C \equiv \widetilde{\mathrm{TQBF}}_n$.

Now suppose that we are working over a finite field $\mathbb{F}$ of characteristic p, and we want to test if a given arithmetic circuit C computes R_n^d for some $d = p^k$. We will employ a similar approach as above, crucially relying on the fact that $(x+y)^{p^k} = x^{p^k} + y^{p^k}$ over $\mathbb{F}$.

Theorem 103. *Let $\mathbb{F}$ be a finite field of characteristic p, and let $d = p^k$ for some $k \geq 0$. Given a* PIT *oracle, one can test in polynomial time if a given arithmetic circuit C computes $\widetilde{\mathrm{TQBF}}_n^d$. Moreover, if the degree of C is at most its size $|C|$, and if $d \leq |C|$, then a* low-PIT *oracle suffices.*

Proof. First observe that $\widetilde{\mathrm{TQBF}}_n^d = (\sum_i z_i \cdot f_i)^d = \sum_i z_i^d \cdot f_i^d$. Let C_i be obtained from C by fixing $z_i = 1$ and $z_j = 0$ for $j \neq i$. Each C_i is a candidate polynomial f_i^d, and we will test that inductively as follows. For $i = m(n)$, we construct $\hat{\theta}_n = f_{m(n)}$, and verify, using the PIT oracle, that $C_{m(n)} \equiv \hat{\theta}_n^d$.

Observe that $q^d(y_1,\ldots,y_t) = q(y_1^d,\ldots,y_t^d)$ for any polynomial q over $\mathbb{F}$. Thus,

$$f_{i-1}^d(a_1,a_2,\ldots,x_1,x_2,\ldots) = f_{i-1}(a_1^d,a_2^d,\ldots,x_1^d,x_2^d,\ldots) = [\mathscr{O}_i f_i](a_1^d,a_2^d,\ldots,x_1^d,x_2^d,\ldots),$$

for some operator $\mathscr{O}_i \in \{\Lambda_{\mathrm{X}_j}, \Lambda_{\mathrm{A}_j}, \exists_{\mathrm{X}_j}, \forall_{\mathrm{X}_j}\}$.

For each operator $\mathscr{O}$, define a new operator $\mathscr{O}^d$ to be $\mathscr{O}$ applied to the dth power of a variable:

$$\begin{aligned}
\forall^d_{\mathrm{Y}_i} q(\vec{y}) &= q\restriction_{y_i \leftarrow 1} \cdot q\restriction_{y_i \leftarrow 0} \\
\exists^d_{\mathrm{Y}_i} q(\vec{y}) &= 1 - (1 - q\restriction_{y_i \leftarrow 1}) \cdot (1 - q\restriction_{y_i \leftarrow 0}) \\
\Lambda^d_{\mathrm{Y}_i} q(\vec{y}) &= y_i^d \cdot q\restriction_{y_i \leftarrow 1} + (1 - y_i^d) \cdot q\restriction_{y_i \leftarrow 0}
\end{aligned}$$

Using the fact that $(x+y)^d = x^d + y^d$ over our field $\mathbb{F}$, one can easily verify that

$$[\mathscr{O}_i f_i](a_1^d, a_2^d, \ldots, x_1^d, x_2^d, \ldots) = \mathscr{O}_i^d [f_i^d(a_1, a_2, \ldots, x_1, x_2, \ldots)].$$

Thus

$$C_{i-1}(\vec{a}, \vec{x}) \equiv f^d_{i-1}(\vec{a}, \vec{x}) \;\Leftrightarrow\; C_{i-1}(\vec{a}, \vec{x}) \equiv \mathscr{O}_i^d [f_i^d(a_1, a_2, \ldots, x_1, x_2, \ldots)].$$

The latter is equivalent, by induction, to the identity $C_{i-1}(\vec{a}, \vec{x}) \equiv \mathscr{O}_i^d [C_i(a_1, a_2, \ldots, x_1, x_2, \ldots)]$, which we can test with the help of our PIT oracle. Finally, we also test if $C \equiv \sum_i z_i \cdot C_i$. If all tests pass, we know that C computes $\widetilde{\mathrm{TQBF}}^d_n$ over $\mathbb{F}$.

For the "moreover" part of the theorem, observe that low-PIT oracle suffices if C has degree at most $|C|$ and if $d \leq |C|$, because then all arithmetic circuits involved in our PIT tests would also have bounded degree. □

6.4.4 Testing arithmetic circuits for equality with $c \cdot \widetilde{\mathrm{TQBF}}_n$

In the case of circuits over the integers, we allow an efficient circuit to compute a constant multiple of some polynomial f. Therefore, we must modify the identity test, similar to the above.

Theorem 104. *Let C be a constant-free arithmetic circuit over $\mathbb{Z}$, and let g be a constant-free, variable-free arithmetic circuit over $\mathbb{Z}$. Thus, g computes a constant, which we also refer to as g.*

Given a PIT *oracle and a pair of such circuits C and g, one can test in polynomial time if C computes $g \cdot \widetilde{\mathrm{TQBF}}_n$. Moreover, if the formal degree of C is at most its size $|C|$, and if the formal degree of g is at most its size $|g|$, then a* low-PIT *oracle suffices.*

Proof. Let the circuits C (computing a function) and g computing a constant (with no input variables) be given as input. We will use identity testing to determine if $g \cdot \widetilde{\mathrm{TQBF}}_n \equiv p(C)$. In the following we will use g to refer to the circuit computing g and the constant value g interchangably.

Let C_i be obtained from C by fixing $z_i = 1$ and $z_j = 0$ for $j \neq i$. Each C_i is a candidate polynomial gf_i, and we will test that inductively as follows. For $i = m(n)$, we construct $\hat{\theta}_n = f_{m(n)}$, and verify, using the PIT oracle, that $C_{m(n)} \equiv g\hat{\theta}_n$.

For each operator, we require a different identity test. The tests are as described above, but taking into account the constant factor by which C and $\widetilde{\mathrm{TQBF}}_n$ will differ.

The first case: $f_i = \Lambda_{\mathrm{Y}_i}[f_{i-1}]$. If C indeed computes $\widetilde{\mathrm{TQBF}}_n$, we will have:

$$\begin{aligned}
C_i &= g \cdot f_i \\
&= g(y_i \cdot f_{i-1}\restriction_{y_i \leftarrow 1} + (1-y_i) \cdot f_{i-1}\restriction_{y_i \leftarrow 0}) && \text{definition of } \widetilde{\mathrm{TQBF}}_n \\
&= y_i \cdot gf_{i-1}\restriction_{y_i \leftarrow 1} + (1-y_i) \cdot gf_{i-1}\restriction_{y_i \leftarrow 0} && \text{distribute}
\end{aligned}$$

Therefore, in this case we should test the identity:

$$C_i \equiv y_i \cdot C_{i-1}\restriction_{y_i \leftarrow 1} + (1-y_i) \cdot C_{i-1}\restriction_{y_i \leftarrow 0}$$

The next case: $f_i = \forall_{\mathrm{X}_i}[f_{i-1}] = f_{i-1}\restriction_{x_i \leftarrow 1} \cdot f_{i-1}\restriction_{x_i \leftarrow 0}$. In this case:

$$
\begin{aligned}
C_i &= g \cdot f_i \\
&= g(f_{i-1}\restriction_{x_i \leftarrow 1} \cdot f_{i-1}\restriction_{x_i \leftarrow 0}) && \text{definition of } \widetilde{\mathrm{TQBF}}_n
\end{aligned}
$$

Multiplying both sides by g and re-arranging we have:

$$
\begin{aligned}
gC_i &= g^2(f_{i-1}\restriction_{x_i \leftarrow 1} \cdot f_{i-1}\restriction_{x_i \leftarrow 0}) \\
&= gf_{i-1}\restriction_{x_i \leftarrow 1} \cdot gf_{i-1}\restriction_{x_i \leftarrow 0}
\end{aligned}
$$

Therefore we should test the identity:

$$
gC_i \equiv C_{i-1}\restriction_{x_i \leftarrow 1} \times C_{i-1}\restriction_{x_i \leftarrow 0}
$$

The last case: $f_i = \exists_{\mathrm{X}_i}[f_{i-1}] = 1 - (1 - f_{i-1}\restriction_{x_i \leftarrow 1}) \cdot (1 - f_{i-1}\restriction_{x_i \leftarrow 0})$. So:

$$
\begin{aligned}
C_i = gf_i &= g(1 - (1 - f_{i-1}\restriction_{x_i \leftarrow 1}) \cdot (1 - f_{i-1}\restriction_{x_i \leftarrow 0})) \\
&= g - g \cdot (1 - f_{i-1}\restriction_{x_i \leftarrow 1}) \cdot (1 - f_{i-1}\restriction_{x_i \leftarrow 0}) \\
&= g - (g - gf_{i-1}\restriction_{x_i \leftarrow 1}) \cdot (1 - f_{i-1}\restriction_{x_i \leftarrow 0})
\end{aligned}
$$

To obtain an identity we can test, we multiply both sides by g:

$$gC_i = g^2 - g \cdot (g - gf_{i-1}\restriction_{x_i \leftarrow 1}) \cdot (1 - f_{i-1}\restriction_{x_i \leftarrow 0})$$
$$= g^2 - (g - gf_{i-1}\restriction_{x_i \leftarrow 1}) \cdot (g - gf_{i-1}\restriction_{x_i \leftarrow 0})$$

Therefore, we should test the identity:

$$gC_i \equiv g^2 - (g - C_{i-1}\restriction_{x_i \leftarrow 1}) \times (g - C_{i-1}\restriction_{x_i \leftarrow 0})$$

Since we have circuits for both C and g, it is easy to construct the circuits in the sqeuence of identity tests. There are $m(n)$ of these tests, which makes the number of calls to the PIT oracle polynomially bounded.

For the "moreover" part of the theorem, observe that a low-PIT oracle suffices if C has formal degree at most $|C|$ and if g has formal degree at most $|g|$ because then all arithmetic circuits involved in our PIT tests would also have bounded degree.

□

6.5 PIT algorithms vs. circuits over finite fields

In this section, we tighten the connections between derandomization of PIT and arithmetic complexity over finite fields.

Theorem 105. *Fix an arbitrary finite field $\mathbb{F}$. We have*

1. $\mathsf{low\text{-}PIT} \in \mathsf{NSUBEXP}_c \Rightarrow \mathsf{ml\text{-}NE} \not\subseteq \mathsf{ASizeDegPow}[\mathsf{superpoly}_c]$.
2. $\mathsf{ml\text{-}NE} \not\subseteq \mathsf{ASizeDegPow}[\mathsf{superpoly}_c] \Rightarrow \mathsf{low\text{-}PIT} \in \mathsf{ro}_\star\text{-}\mathsf{NSUBEXP}_c$.

Proof sketch. (1) Assume a nondeterministic subexponential-time algorithm for low-PIT, but that ml-NE has "small" arithmetic circuits. We arithmetize TQBF to get a PSPACE-complete

multilinear polynomial $f = \{f_n\}$ over $\mathbb{F}$. This polynomial f turns out to be computable in ml-NE, and so, by our assumption, some powers $f_n^{d_n}$, for "small" d_n, have small arithmetic circuits over $\mathbb{F}$.

For each n, we nondeterministically guess a small circuit and a small d_n. Using the ideas of the PSPACE = IP proof, we then verify that the guessed circuit computes the polynomial $f_n^{d_n}$. This verification algorithm uses our assumed low-PIT algorithm, and runs in $\mathsf{NSUBEXP}_c$.

Computing powers $f_n^{d_n}$ is still PSPACE-complete, and so we get $\mathsf{PSPACE} \subseteq \mathsf{NSUBEXP}_c$. By padding, we obtain $\mathsf{SPACE}[\mathsf{superpoly}_c] \subseteq \mathsf{NE}$. By diagonalization, $\mathsf{SPACE}[\mathsf{superpoly}_c]$ contains a language L of some computably superpolynomial Boolean circuit complexity, almost everywhere. It follows that the multilinear extension of L over $\mathbb{F}$ requires arithmetic circuits of computably superpolynomial size, almost everywhere. On the other hand, each coefficient of this multilinear polynomial is computable in $\mathsf{SPACE}[\mathsf{superpoly}_c]$, and hence in NE.

(2) Assume that some family $g = \{g_n\}$ of polynomials in ml-NE is such that all powers $g_n^{d_n}$, for "small" d_n, require superpolynomial arithmetic circuit complexity for infinitely many input lengths n. By [KI04], we get that low-PIT is in $\mathsf{NSUBEXP}_c$ infinitely often. The input lengths where low-PIT is easy (derandomized) correspond to the (smaller) input lengths where the polynomials g_n are actually hard.

To improve this "infinitely often" result to the "robustly often" one, we do the following: when asked to derandomize low-PIT for a certain input length n, we go to the related smaller length n', and consider polynomials over a superpolynomial interval of input lengths above n' as candidate hard polynomials; we use each such polynomial to construct a candidate hitting set by [KI04]. If the given interval above n' contains a length m such that g_m is hard, then our derandomization succeeds. Since there infinitely many intervals containing a length m where g_m is hard, there will be infinitely many intervals of superpolynomial length where our low-PIT algorithm is correct. □

We now lift Theorem 105 to an equivalence, using robustness.

Theorem 106. *Fix an arbitrary finite field $\mathbb{F}$. We have*

$$\mathsf{low\text{-}PIT} \in \mathsf{ro}_\star\text{-}\mathsf{NSUBEXP}_c \;\Leftrightarrow\; \mathsf{rp}_\star\text{-}\mathsf{ml\text{-}NE} \not\subseteq \mathsf{ASizeDegPow}[\mathsf{superpoly}_c].$$

Proof sketch. ($\Rightarrow$) We start as in the proof of Theorem 105, implication (1). We get a PSPACE-complete multilinear polynomial $f = \{f_n\}$ over $\mathbb{F}$ such that some powers $f_n^{d_n}$ are computable by small arithmetic circuits, for almost all input lengths n. Since our low-PIT algorithm is correct for infinitely many superpolynomial intervals of input lengths, we can guarantee the successful verification of an arithmetic circuit for $f_n^{d_n}$ for the corresponding superpolynomial intervals of input lengths only. This yields $\mathsf{PSPACE} \subseteq \mathsf{ro}_\star\text{-}\mathsf{NSUBEXP}_c$, and by padding, $\mathsf{SPACE}[\mathsf{superpoly}_c] \subseteq \mathsf{ro}_\star\text{-}\mathsf{NE}$. Finally, by diagonalization and multilinear extension, we get a family of multilinear polynomals g_n over $\mathbb{F}$ that require computably superpolynomial arithmetic circuit complexity almost everywhere, and yet we can compute the coefficients of g_n in NE for infinitely many superpolynomial intervals of input sizes n.

($\Leftarrow$) As in the proof of Theorem 105, implication (2), we will use hard polynomials to derandomize low-PIT by [KI04]. The difference now is that a given NE machine computes a valid polynomial only over some computably robust set S of input lengths n, and that this polynomial is hard only for infinitely many lengths $n \in \mathsf{core}(S)$. Still we can use this NE machine to construct a candidate hitting set for a given low-PIT instance so that, for infinitely many lengths $n \in \mathsf{core}(S)$, we get a correct hitting set, and so $\mathsf{low\text{-}PIT} \subseteq \mathsf{io}\text{-}\mathsf{NSUBEXP}_c$. To boost this to the robustly often inclusion, we employ a similar trick as before: use a superpolynomial-size interval of input lengths to get a collection of candidate hitting sets, and take their union. When all input lengths fall into an interval where our NE machine computes a valid polynomial, we get that the union of such candidate hitting sets is well-defined. If, in addition, the polynomial computed by our NE machine is actually hard on some length in this interval, then we get a correct hitting set. □

In the remaining sub-sections, we give detailed proofs of Theorems 105 and 106 above.

6.5.1 Proof of implication (1) of Theorem 105

We argue by contradiction. Suppose there is a low-PIT algorithm running in nondeterministic time $2^{n^{1/\beta(n)}}$, for some $\beta(n) \in \omega_c(1)$. But, ml-NE $\subseteq$ ASizeDegPow$[n^{\alpha(n)}]$ for every $\alpha(n) \in \omega_c(1)$.

Our proof consists of the following three major steps: (i) collapsing PSPACE into NSUBEXP$_c$, (ii) using diagonalization to get hard multilinear polynomials in SPACE[superpoly$_c$], and (iii) using a padding argument to argue that this hard polynomial is actually computable in ml-NE, contradicting the assumption that ml-NE contains no hard polynomials.

We provide the details for these three steps in the following three subsections.

Placing PSPACE into NSUBEXP$_c$

Recall our PSPACE-hard multilinear polynomial $\widetilde{\text{TQBF}}_n$ over $\mathbb{F}$ from Section 6.4. By Lemma 102, MONOMIAL$(\widetilde{\text{TQBF}}_n) \subseteq$ *LINSPACE*, which is in E. Thus, $\widetilde{\text{TQBF}}_n \in$ ml-NE, and so, by assumption, $\widetilde{\text{TQBF}}_n \in$ ASizeDegPow$[s(n)]$ for $s(n) = n^{\alpha(n)}$, for every computably superpolynomial size function $s(n)$. With foresight, let us set $\alpha(n) := \beta^{1/3}(n)$.

It follows that some powers $\widetilde{\text{TQBF}}_n^{d_n}$, for $d_n \leq s(n)$, have arithmetic circuits of size at most $s(n)$ over $\mathbb{F}$. By Lemma 88, there is $d_n' = p^k \leq d_n$, where p is the prime characteristic of the field $\mathbb{F}$, such that $\widetilde{\text{TQBF}}_n^{d_n'}$ is computable by an arithmetic circuit of size at most $s'(n) = (s(n))^c$, for some $c > 0$. By standard arguments, we may assume that the circuit has degree at most that of $\widetilde{\text{TQBF}}_n$[3].

For each n, we nondeterministically guess a circuit C_n of size and degree at most $s'(n)$, and a number $d_n \leq s(n)$. By Theorem 103, we can test if $C_n \equiv \widetilde{\text{TQBF}}_n^{d_n}$ over $\mathbb{F}$ in NSUBEXP$_c$, using the low-PIT algorithm. Running this algorithm on our guessed instances of size $s'(n)$ will take time at most $2^{n^{c\alpha(n)/\beta(n)}}$, which is computably subexponential for our choice of $\alpha(n) = \beta^{1/3}(n)$.

[3]The standard transformation to make a given circuit compute a polynomial of degree at most d is to split the polynomial computed by every gate of the circuit into d homogeneous polynomials of degree i, for $i = 0, \ldots, d$, and keep track of each such homogeneous polynomial by introducing d copies of each gate. This makes the new circuit size polynomial in the old circuit size. Since we are working over a finite field, we don't need to worry about constants getting too big during this transformation of the original circuit.

It follows that we can solve TQBF in $\mathsf{NSUBEXP}_c$, with the running time $\mathsf{poly}(2^{n^{c/\alpha^2(n)}})$.

Diagonalization

Next, consider a language $L \in \mathsf{SPACE}[n^{\alpha^{1.9}(n)}]$ that is not in io-$\mathsf{SIZE}[n^{\alpha^{1.8}(n)}]$. Such a language exists by a standard diagonalization argument [Kan82]: find a circuit of size $n^{\alpha^{1.85}(n)}$ whose Boolean function cannot be computed by any circuit of size at most $n^{\alpha^{1.8}(n)}$; such a circuit exists by a counting argument (see, e.g., [PW86]). A "brute-force" algorithm to find such a circuit needs enough space to store a circuit of size $n^{\alpha^{1.85}(n)}$, a circuit of size at most $n^{\alpha^{1.8}(n)}$, and an input $x \in \{0,1\}^n$, as well as enough space to evaluate these two circuits on x. The algorithm will try every large-size circuit, and check that it differs from every small-size circuit on at least one input of length n.

Let f_L be the multilinear extension of L over the field $\mathbb{F}$. Since an arithmetic circuit over $\mathbb{F}$ can be easily simulated by a Boolean circuit of only polynomially larger size, we get that every power f_L^D of f_L requires arithmetic circuits of size at least $n^{\alpha^{1.7}(n)}$. Thus, in particular, we get $f_L \notin \mathsf{ASizeDegPow}[n^{\alpha(n)}]$.

Padding

Finally, we will argue that $M := \textsc{monomial}(f_L) \in \mathsf{NE}$, contradicting our assumption that ml-$\mathsf{NE} \subseteq \mathsf{ASizeDegPow}[n^{\alpha(n)}]$. Since $L \in \mathsf{SPACE}[n^{\alpha^{1.9}(n)}]$, we get that $M \in \mathsf{SPACE}[n^{\alpha^{1.9}(n)}]$ as well. Consider the padded language $M_{pad} = \{(x, 1^{|x|^{\alpha^{1.9}(|x|)}}) \mid x \in M\}$. Since $\alpha(n)$ is computable, we get that $M_{pad} \in \mathit{LINSPACE}$; this is the place where we use the computability of our time bounds. Using the fact that TQBF $\in \mathsf{NTIME}[\mathsf{poly}(2^{n^{c/\alpha^2(n)}})]$ and that M_{pad} is polynomial-time reducible to TQBF, we get that $M \in \mathsf{NE}$.

6.5.2 Proof of implication (2) of Theorem 105

Assume that some family $g = \{g_n\}$ of polynomials in ml-NE is such that, for some computably superpolynomial function $s(n) = n^{\alpha(n)}$, there are infinitely many input lengths $n \in \mathbb{N}$

where $g_n^{d_n}$ cannot be computed by an arithmetic circuit of size $s(n)$, for any $1 \leq d_n \leq s(n)$.

We will use Theorem 92 to get derandomization of low-PIT. We present our argument in the following subsections.

Evaluating a hard polynomial over enough points

For every n, let $\mathbb{F}'$ be an extension field of $\mathbb{F}$ of size $O(s)$. This extension field can be constructed efficiently, in time $\mathsf{poly}(s)$, by trying all possible polynomials over $\mathbb{F}$ of degree $O(\log s)$, until one finds an irreducible polynomial over $\mathbb{F}$.

Since $\text{MONOMIAL}(g_n) \in \mathsf{NE}$, we can compute the coefficients of all 2^n monomials of g_n in nondeterministic time $\mathsf{poly}(2^n)$. Once we have the coefficients, we can evaluate g_n on any given n-tuple of elements from the extension field $\mathbb{F}'$, in time $\mathsf{poly}(2^n)$. It follows that we can evaluate g_n over all points in $(\mathbb{F}')^n$ in nondeterministic time $\mathsf{poly}(2^n)$.

Building a hitting set

Using Theorem 92, we get, for some constant $c > 0$, a set of $t = s(n)^{n^c}$ of m-tuples from $(\mathbb{F}')^m$, for $m = s^{1/c}$, that is a hitting set for low-PIT instances of size m, provided that n is the length where $g_n^{d_n}$ requires circuit size $s(n)$ for every $1 \leq d_n \leq s(n)$. For infinitely many input lengths n, we will get a valid hitting set of size t for low-PIT instances of size m.

Running time analysis:

We have $t = s(n)^{n^c} = n^{\alpha(n)\cdot n^c} \leq 2^{n^{c+2}}$, where the last inequality is because $s(n) \leq 2^n$. Using $n = m^{c/\alpha(n)}$, we get that $t \leq \exp(m^{d/\alpha(n)})$, for $d = c(c+2)$.

Next we lowerbound $\alpha(n)$. By Lemma 91, we may assume that $\alpha(n)$ satisfies the inequality $\alpha(n^k) \leq \alpha(n) + \lceil \log k \rceil$. Using this as well as the fact that $n \leq m$, we have for large enough m:

$$\alpha(n) = \alpha(m^{c/\alpha(n)}) \geq \alpha(m) - \lceil \log \alpha(n)/c \rceil \geq \alpha(m) - \lceil \log \alpha(m) \rceil \geq \alpha(m)/2.$$

Putting it all together, we get $t \leq \exp(m^{(2d)/\alpha(m)})$, which is computably subexponential.

This yields low-PIT $\in$ io-NSUBEXP$_c$. Next we show how to improve this inclusion from "infinitely often" to "robustly often".

Derandomizing low-PIT robustly often

When asked to derandomize low-PIT for a certain input length m, we go to the smallest length n such that $s(n) \geq m^c$, and consider polynomials $g_{n'}$ over all lengths n' such that $n \leq n' \leq n^{\sqrt{\alpha(n)}}$ as candidate hard polynomials. We use each such polynomial to construct a candidate hitting set via Theorem 92. If the given interval above n contains a length n' such that $g_{n'}$ is hard, then our derandomization succeeds. Since there infinitely many intervals containing a length n' where $g_{n'}$ is hard, there will be infinitely many intervals of superpolynomial length where our low-PIT algorithm is correct.

It remains to analyze the running time of the described algorithm. It will be dominated by the value t in Theorem 92 for $n' = n^{\sqrt{\alpha(n)}}$. We get $t \leq (m^c)^{(n')^c} \leq \exp(n^{(c+2)\sqrt{\alpha(n)}}) \leq \exp(m^{c(c+2)/\sqrt{\alpha(n)}})$. Using our earlier lower bound on α of $\alpha(n) \geq \alpha(m)/2$, we have $t \leq \exp(m^{c'/\sqrt{\alpha(m)}})$, for $c' = c(c+2)\sqrt{2}$, and so low-PIT $\in$ ro$_\star$-NSUBEXP$_c$.

6.5.3 Robust derandomization of PIT$_\mathbb{F}$ implies robust circuit lower bounds over $\mathbb{F}$ (forward direction of Theorem 106)

Again, we argue by contradiction. Suppose there is a low-PIT algorithm running in nondeterministic time $2^{n^{1/\beta(n)}}$, for some $\beta(n) \in \omega_c(1)$, on a ρ-robust set S for some $\rho(n) \in \omega_c(1)$. But, rp$_\star$-ml-NE $\subseteq$ ASizeDegPow$[n^{\alpha(n)}]$ for every $\alpha(n) \in \omega_c(1)$.

First, we may assume, without loss of generality, that $\beta(n) = \rho(n)$, and moreover, for every $k \geq 1$, $\beta(n^k) \leq \beta(n) + \lceil \log k \rceil$. Indeed, if not, define $\delta'(n) := \min\{\beta(n), \rho(n)\}$, and use Lemma 91 to get $\delta(n) \leq \delta'(n)$ satisfying the property $\delta(n^k) \leq \delta(n) + \lceil \log k \rceil$ for all $k \geq 1$. It is easy to see that the δ-intervals around each $n \in \mathsf{core}(S)$ are contained within S, since $\delta(n) \leq \rho(n)$; so we get a δ-robust subset $S' \subseteq S$. Also, since $\delta(n) \leq \beta(n)$, we have $2^{n^{1/\beta(n)}} \leq 2^{n^{1/\delta(n)}}$. Hence,

we get a nondeterministic $2^{n^{1/\delta(n)}}$-time algorithm deciding low-PIT on a δ-robust set S' of input lengths.

Next, as in the proof of Theorem 105, implication (1), our proof consists of the following three major steps: (i) collapsing PSPACE into $\mathsf{ro}_\star$-NSUBEXP$_c$, (ii) using diagonalization to define a hard multilinear polynomial is SPACE[superpoly$_c$], and (iii) using a padding argument to argue that this hard polynomial is actually computable in $\mathsf{rp}_\star$-ml-NE, contradicting the assumption that $\mathsf{rp}_\star$-ml-NE contains no hard polynomials.

Placing PSPACE **into** $\mathsf{ro}_\star$-NSUBEXP$_c$

As before, our PSPACE-hard multilinear polynomial $\widetilde{\mathrm{TQBF}}_n$ over $\mathbb{F}$ is computable in ml-E, and so, by assumption, $\widetilde{\mathrm{TQBF}}_n \in \mathsf{ASizeDegPow}[s(n)]$ for $s(n) = n^{\alpha(n)}$, for every $\alpha(n) \in \omega_c(1)$. By Lemma 88, there is $d'_n = p^k \leq s(n)$, where p is the prime characteristic of the field $\mathbb{F}$, such that $\widetilde{\mathrm{TQBF}}_n^{d'_n}$ is computable by an arithmetic circuit of size and degree at most $s'(n) = (s(n))^c$, for some $c > 0$.

For each n, we nondeterministically guess a circuit C_n of size and degree at most $s'(n)$, and a number $d_n \leq s(n)$. By Theorem 103, we can test if $C_n \equiv \widetilde{\mathrm{TQBF}}_n^{d_n}$ over $\mathbb{F}$, using the low-PIT algorithm. Running this algorithm on our guessed instances of size $s'(n)$ will take time at most $2^{n^{c\alpha(n)/\beta(n)}}$.

Since the reduction from checking $C_n \equiv \widetilde{\mathrm{TQBF}}_n^{d_n}$ to low-PIT is polynomial-time honest Turing (Theorem 103) and low-PIT is paddable, we get that TQBF is reducible to low-PIT via an honest Turing reduction running in time $t(n) = (s'(n))^{c'} = n^{cc'\alpha(n)}$, for some constant $c' > 0$. Let $\gamma(n) := \sqrt{\beta(n)}/2$ and $\alpha(n) := \gamma(n)/(cc')$. We get by Lemma 96 that TQBF is decided by a nondeterministic $2^{n^{1/\gamma(n)}}$-time algorithm over a γ-robust set of input lengths.

Diagonalization

For every $\tau(n) \in \omega_c(1)$, there is a language $L \in \mathsf{SPACE}[n^{\tau(n)}] \setminus \mathsf{io\text{-}SIZE}[n^{\tau^{0.8}(n)}]$ (due to [Kan82]). Let f_L be the multilinear extension of L over the field $\mathbb{F}$. Since an arithmetic circuit

over $\mathbb{F}$ can be easily simulated by a Boolean circuit of only polynomially larger size, we get that every power f_L^D of f_L requires arithmetic circuits of size at least $n^{\tau^{0.5}(n)}$. Thus we get $f_L \notin \mathsf{ASizeDegPow}[n^{\tau^{0.5}(n)}]$.

Padding

For $\tau(n) \in \omega_c(1)$ to be determined, consider the hard language L and the hard polynomial f_L as defined above. We will argue that $M := \textsc{monomial}(f_L) \in \mathsf{ro}_\star\text{-}\mathsf{NE}$. This will imply that $f_L \in \mathsf{rp}_\star\text{-ml-}\mathsf{NE}$, thereby contradicting our assumption that $\mathsf{rp}_\star\text{-ml-}\mathsf{NE} \subseteq \mathsf{ASizeDegPow}[n^{\alpha(n)}]$ for every $\alpha(n) \in \omega_c(1)$.

Since $L \in \mathsf{SPACE}[n^{\tau(n)}]$, we get that $M \in \mathsf{SPACE}[n^{\tau(n)}]$ as well. Consider the padded language $M_{pad} = \{(x, 1^{|x|^{\tau(|x|)}}) \mid x \in M\}$. Since $\tau(n)$ is computable, we get that $M_{pad} \in \mathit{LINSPACE}$.

We have that M_{pad} is reducible to TQBF by an honest polynomial-time Turing reduction, and that M is reducible to M_{pad} via an honest mapping reduction in time $n^{\tau(n)}$. It follows that M is reducible to TQBF via an honest Turing reduction in time $n^{c''\tau(n)}$ for some constant $c'' > 0$.

Recall that TQBF is decided by a nondeterministic $2^{n^{1/\gamma(n)}}$-time algorithm over a γ-robust set of input lengths. Set $\gamma'(n) = \min\{\gamma(n), \gamma'(\lfloor\sqrt{n}\rfloor) + 1\}$, so that by Lemma 91, we have for every $k \geq 1$ that $\gamma'(n^k) \leq \gamma'(n) + \lceil \log k \rceil$. Set $\gamma''(n) := \sqrt{\gamma'(n)}/2$ and set $\tau(n) := \gamma''(n)/c''$. We get by Lemma 96 that M is decided by a nondeterministic $2^{n^{1/\gamma''(n)}}$-time algorithm over a γ''-robust set of input lengths. It follows that $\mathsf{rp}_\star\text{-ml-}\mathsf{NE} \not\subseteq \mathsf{ASizeDegPow}[n^{\tau^{0.5}(n)}]$. But $\tau^{0.5}(n) \in \omega_c(1)$, and so we contradict the assumption that $\mathsf{rp}_\star\text{-ml-}\mathsf{NE} \subseteq \mathsf{ASizeDegPow}[n^{\alpha(n)}]$ for every $\alpha(n) \in \omega_c(1)$.

6.5.4 Robust circuit lower bounds over $\mathbb{F}$ imply robust derandomization of $\mathsf{PIT}_\mathbb{F}$ (backwards direction of Theorem 106)

Assume that there is some family $g = \{g_n\}$ of promise polynomials in $\mathsf{rp}_\star\text{-ml-}\mathsf{NE}$ on computably robust set S such that, for some computably superpolynomial function $s(n) = n^{\alpha(n)}$, there are infinitely many input lengths $n \in \mathsf{core}(S)$ where $g_n^{d_n}$ cannot be computed by an arithmetic circuit of size $s(n)$, for any $1 \leq d_n \leq s(n)$.

As in the proof of Theorem 105, we will use Theorem 92 to get derandomization of low-PIT. We present our argument in the following subsections. Similar to the proof of Theorem 105, we search an interval to obtain derandomization robustly often. The only difference is that we are not free to set the size of the interval that we search.

Building a hitting set by searching intervals

Let G be the "good" set of input lengths obtained by applying Lemma 97 to S. Recall, that G is exactly the set of input lengths n where an interval around a $n^{o_c(1)}$ function of n is guaranteed to contain a hard length for g. Suppose we are given a low-PIT instance of size m, for $m \in G$. Then, we use the following algorithm:

1. Non-deterministically construct the truth table of $g_{m'}$, for each $m' \in I[m]$.
2. Construct a HSG from each $g_{m'}$ (as in Theorem 105, we may need to work over an extension field of $\mathbb{F}$ but this is tractable).
3. Test the circuit on the output of each HSG.

Step 1 will succeed on some path if each $m' \in I[m]$ is also in S, which we have by definition. Step 2 will succeed (the HSG works) when $m = s^{1/c}$ for a given constant c and hardness parameter s. We know that there exists $m' \in I[m]$ such that $f_{m'} \notin \mathsf{ASizeDegPow}[m'^{\alpha(m')}]$, and we can pad low-PIT, so deciding PIT on instances larger than m suffices to decide PIT_m. Therefore, if $m < (m')^{\alpha(m')}$ for all $m' \in I[m]$, then the algorithm will work. But $(m')^{\alpha(m')}$ is minimized at exactly $T_l(m)^{\alpha(T_l(m))}$, which by Lemma 97 is still $m^{\omega_c(1)}$, and so always exceeds m. Therefore, this algorithm is correct for any input length in G.

As in Theorem 105, the runtime is dominated by t, the maximal number of tuples that we need to evaluate to run a HSG. Write $t = 2^{((m')^c)^{\log(s)}} < 2^{(m')^{c+1}}$, which is maximized at $m' = T_h(m)$. But by Lemma 97, $T_l(n)^{\alpha(T_l(n))} \in n^{\omega_c(1)}$. This, along with the fact that G is computably robust, places $\mathsf{low\text{-}PIT} \in \mathsf{ro}_\star\text{-}\mathsf{NSUBEXP}_c$.

6.6 PIT algorithms vs. circuits over the integers

We have analogous results also for the case of integers $\mathbb{Z}$, with analogous proofs (using the analysis of [JS12] showing that Kaltofen's [Kal89] polynomial factorization algorithm over integers respects formal degree).

Theorem 107. *Over $\mathbb{Z}$, we have*

1. $\mathsf{low\text{-}PIT} \in \mathsf{NSUBEXP}_c \Rightarrow \mathsf{ml\text{-}NE} \not\subseteq \mathsf{ASizeDegMul}[\mathsf{superpoly}_c]$.

2. $\mathsf{ml\text{-}NE} \not\subseteq \mathsf{ASizeDegMul}[\mathsf{superpoly}_c] \Rightarrow \mathsf{low\text{-}PIT} \in \mathsf{ro}_\star\text{-}\mathsf{NSUBEXP}_c$.

Theorem 108. *Over $\mathbb{Z}$,*

$$\mathsf{low\text{-}PIT} \in \mathsf{ro}_\star\text{-}\mathsf{NSUBEXP}_c \Leftrightarrow \mathsf{rp}_\star\text{-}\mathsf{ml\text{-}NE} \not\subseteq \mathsf{ASizeDegMul}[\mathsf{superpoly}_c].$$

Sketched proof of implication (1) of Theorem 107

As in the proof of Theorem 105, we follow three major steps: (i) collapsing PSPACE into $\mathsf{NSUBEXP}_c$, (ii) diagonalization to get a hard multilinear polynomial in $\mathsf{SPACE}[\mathsf{superpoly}_c]$, and (iii) using a padding argument to argue that this hard polynomial is actually computable in ml-NE, contradicting the assumption that ml-NE contains no hard polynomials.

Identical to the proof of Theorem 105, except for the following two changes to step (i) where we cause the collapse:

- Where we would attempt to find a $\widetilde{\text{TQBF}}$ circuit by guessing a single $s(n)$ size circuit over a finite field, we instead guess two $s(n)$-size circuits over the integers, one variable-free. Then, instead of invoking Theorem 103, we invoke Theorem 104 to determine if one of these circuits computes $c \times \widetilde{\text{TQBF}}$. The runtime analysis is identical.

- In the finite field case, it is obvious that we can evaluate these circuits in time $\mathsf{poly}(s(n))$ to actually compute $\widetilde{\text{TQBF}}_n$, because the bit-size of all intermediate values is always

constant for a particular finite field.. In the integer case, we need the additional fact that $\mathsf{ASizeDegMul}[s(n)]$ circuits have bounded formal degree and are constant-free. This implies that the bit-sizes of the intermediate values computes by ASizeDegMul circuits are bounded, and so we can in fact evaluate them efficiently in $\mathsf{NSUBEXP}_c$.

Sketched proof of implication (2) of Theorem 107

This direction is identical to the proof of Theorem 105, except that it uses the statement of Theorem 92 over $\mathbb{Z}$ from [JS12] and never needs to pass to an extension field. Again, we use that since ASizeDegMul circuits are constant-free and of bounded formal degree, the bit-sizes of intermediate values are bounded and they can be evaluated efficiently.

Finally observe that the proof of the robust equivalences for the case of the integers, Theorem 108, is identical to the proof of the finite equivalence, Theorem 106, save for the same changes that are required to adapt the proof of implications for the finite case (Theorem 105) to implications for the integers (Theorem 107), which we just covered above.

6.7 Promise-BPP vs. Boolean circuit lower bounds

Theorem 109. *We have*

1. $\mathsf{promise\text{-}BPP} \subseteq \mathsf{NSUBEXP}_c \Rightarrow (\mathsf{NE} \cap \mathsf{coNE}) \not\subseteq \mathsf{SIZE}[\mathsf{superpoly}_c]$.

2. $(\mathsf{NE} \cap \mathsf{coNE}) \not\subseteq \mathsf{SIZE}[\mathsf{superpoly}_c] \Rightarrow \mathsf{promise\text{-}BPP} \subseteq \mathsf{ro}_\star\text{-}\mathsf{NSUBEXP}_c$.

Proof sketch. (1) Assume that CAPP is correctly approximated by an $\mathsf{NSUBEXP}_c$ algorithm, but every language in $(\mathsf{NE} \cap \mathsf{coNE})$ has small superpolynomial-size Boolean circuits. The latter means that $\mathsf{E} \subseteq \mathsf{SIZE}[\mathsf{superpoly}_c]$, and hence, by [BFL91], that $\mathsf{E} \subseteq \mathsf{MA}[\mathsf{superpoly}_c]$, for a related small superpolynomial time complexity. Using our CAPP algorithm, we get $\mathsf{MA}[\mathsf{superpoly}_c] \subseteq \mathsf{NSUBEXP}_c$. By padding the inclusion $\mathsf{E} \subseteq \mathsf{NSUBEXP}_c$, we get that $\mathsf{TIME}[2^{\mathsf{superpoly}_c}] \subseteq \mathsf{NE}$. Finally, by diagonalization, we get a language $L \in \mathsf{TIME}[2^{\mathsf{superpoly}_c}] \subseteq \mathsf{NE}$ that requires computably

superpolynomial circuit complexity (almost everywhere). Since $\bar{L} \in \mathsf{TIME}[2^{\mathsf{superpoly}_c}] \subseteq \mathsf{NE}$, the conclusion follows.

(2) Assume we have a pair of NE machines that compute L and $\bar{L}$ for a language L requiring $\mathsf{superpoly}_c$-size circuits infinitely often. By the hardness-randomness tradeoff of [BFNW93], we get a $\mathsf{NSUBEXP}_c$ algorithm that correctly approximates CAPP infinitely often; the input lengths where the CAPP algorithm is correct correspond to the (smaller) input lengths where the language L is actually hard.

To boost this to the desired $\mathsf{promise\text{-}BPP} \subseteq \mathsf{ro}_\star\text{-}\mathsf{NSUBEXP}_c$ inclusion, we use the same "interval trick" as in the arithmetic case (Theorem 105, (2)), to show that there is a $\mathsf{NSUBEXP}_c$ algorithm that, robustly often, produces a discrepancy set for circuits of size n. Indeed, on input length n, take the corresponding input length n' where Theorem 93 requires $L_{n'}$ to be hard in order to build a discrepancy set for circuits of size n. Consider L_m for a superpolynomial interval of lengths $m \geq n'$, and concatenate the truth tables of all these L_m's. This new string must be hard as it contains a hard substring. Applying Theorem 93 to this hard string, we get a discrepancy set. □

We now extend the two implications of Theorem 109 to the equivalence below, by carefully adapting the arguments above to the setting of robust inclusions and separations. The proof will be quite similar to that of Theorem 109, except we will need to carefully maintain the condition that all operations which change the size of an input preserve membership in robust sets. Thus, many steps of this argument will be shared with the poof of Theorem 106 about the finite field setting, and rely on the same lemmas about closure properties of computably robust sets.

Theorem 110. $\mathsf{promise\text{-}BPP} \subseteq \mathsf{ro}_\star\text{-}\mathsf{NSUBEXP}_c \Leftrightarrow \mathsf{rp}_\star\text{-}(\mathsf{NE} \cap \mathsf{coNE}) \not\subseteq \mathsf{SIZE}[\mathsf{superpoly}_c]$.

Proof sketch. ($\Rightarrow$) Assume promise-BPP is easy, but no hard language is in $\mathsf{rp}_\star\text{-}(\mathsf{NE} \cap \mathsf{coNE})$. As in the proof of Theorem 109, implication (1), we conclude that $\mathsf{E} \subseteq \mathsf{SIZE}[\mathsf{superpoly}_c]$ and so $\mathsf{E} \subseteq \mathsf{MA}[\mathsf{superpoly}_c]$. Under our assumption on promise-BPP, we can derandomize $\mathsf{MA}[\mathsf{superpoly}_c]$

only robustly often, getting that $\mathsf{E} \subseteq \mathsf{ro}_\star\text{-}\mathsf{NSUBEXP}_c$. By padding, $\mathsf{TIME}[2^{\mathsf{superpoly}_c}] \subseteq \mathsf{ro}_\star\text{-}\mathsf{NE}$, and by diagonalization, $\mathsf{TIME}[2^{\mathsf{superpoly}_c}]$ contains a language L that requires $\mathsf{superpoly}_c$ circuit size almost everywhere. Since the language $\{(x,1) \mid x \in L\} \cup \{(x,0) \mid x \in \bar{L}\}$ is also in $\mathsf{TIME}[2^{\mathsf{superpoly}_c}] \subseteq \mathsf{ro}_\star\text{-}\mathsf{NE}$, we get a pair of NE machines that correctly decide L_n and $\overline{L_n}$, respectively, on a computably robust set of input lengths n.

($\Leftarrow$) Assume we have a pair of NE machines that correctly compute L and $\bar{L}$ for some computably robust set S of input lengths, where L requires $\mathsf{superpoly}_c$ circuit size for infinitely many input lengths in $\mathsf{core}(S)$. Using these machines, we nondeterministically guess for each n a candidate hard function on n bits, in time $\mathsf{poly}(2^n)$. By our assumption, infinitely often, we get a truly hard function. So we get, using Theorem 93, a correct discrepancy set for circuits of size n, for infinitely many n, computable in $\mathsf{NSUBEXP}_c$. Finally, as in the proof of Theorem 109, implication (2), we use the "interval trick" to conclude the argument. □

In the remaining sub-sections, we give detailed proofs of Theorems 109 and 110 above.

6.7.1 Proof of implication (1) of Theorem 109

Assume that CAPP is correctly approximated by a nondeterministic algorithm running in time $2^{n^{1/\beta(n)}}$, for some $\beta(n) \in \omega_c(1)$, but $(\mathsf{NE} \cap \mathsf{coNE}) \subseteq \mathsf{SIZE}[n^{\alpha(n)}]$ for every $\alpha(n) \in \omega_c(1)$.

Collapsing E to NSUBEXP$_c$

The latter assumption implies that $\mathsf{E} \subseteq \mathsf{SIZE}[n^{\alpha(n)}]$, and so, by [BFL91], that $\mathsf{E} \subseteq \mathsf{MA}[n^{c\alpha(n)}]$ for some constant $c > 0$. Using our CAPP algorithm, we derandomize the MA class, getting $\mathsf{MA}[n^{c\alpha(n)}] \subseteq \mathsf{NTIME}[2^{n^{c\alpha(n)/\beta(n)}}]$. By setting $\alpha(n) := (\beta(n)/c)^{1/3}$, the latter becomes computably subexponential nondeterministic time $\exp(n^{1/\alpha^2(n)})$. It follows that $\mathsf{E} \subseteq \mathsf{NTIME}\left[2^{n^{1/\alpha^2(n)}}\right]$.

Diagonalization

By standard diagonalization, we get a language $L \in \mathsf{TIME}\left[2^{n^{\alpha^2(n)}}\right] \notin \text{io-}\mathsf{SIZE}[n^{\alpha(n)}]$ as follows. By enumeration, find an n-input Boolean circuit of size $n^{\alpha^{1.5}(n)}$ that differs from every n-input Boolean circuit of size at most $n^{\alpha(n)}$ on at least one input $x \in \{0,1\}^n$; such a circuit exists by a counting argument [PW86]. The running time is easily seen to be at most $2^{n^{\alpha^2(n)}}$.

Next we show that this hard language L is in $\mathsf{NE} \cap \mathsf{coNE}$, contradicting our assumption that every language in $(\mathsf{NE} \cap \mathsf{coNE})$ has circuits of size $[n^{\alpha(n)}]$.

Padding

For the hard language L defined above, consider its graph version $M = \{(x,0) \mid x \notin L\} \cup \{(x,1) \mid x \in L\}$. Since L is computable in deterministic time $2^{n^{\alpha^2(n)}}$, we conclude that M is computable within the same time bounds. Define the padded version $M_{pad} = \{(x, 1^{|x|^{\alpha^2(|x|)} - |x|}) \mid x \in M\}$ of M. Since $\alpha(n)$ is efficiently computable, we get that $M_{pad} \in \mathsf{E}$. Using $\mathsf{E} \subseteq \mathsf{NTIME}[2^{n^{1/\alpha^2(n)}}]$, we conclude that $M \in \mathsf{NE}$. The latter implies that $L \in (\mathsf{NE} \cap \mathsf{coNE})$. Hence, $(\mathsf{NE} \cap \mathsf{coNE}) \not\subseteq \mathsf{SIZE}[n^{\alpha(n)}]$. A contradiction.

6.7.2 Proof of implication (2) of Theorem 109

Assume we have a pair of NE machines that compute L and $\bar{L}$ for a language L requiring $s(n)$-size circuits infinitely often, for some computably superpolynomial $s(n) = n^{\alpha(n)}$.

Our proof structure is similar to that of Theorem 105, and is given in the following subsections.

Constructing the truth table of a hard function

For every n, we use our NE machines for L and $\bar{L}$ to construct the truth table of L_n in nondeterministic time $\mathsf{poly}(2^n)$ as follows. For each $x \in \{0,1\}^n$, nondeterministically guess $b_x = 0$ or $b_x = 1$. If $b_x = 0$, guess a witness for $x \notin L$. If $b_x = 1$, guess a witness for $x \in L$. Verify the correctness of all guessed witnesses over all $x \in \{0,1\}^n$ using the pair of NE machines for L

and $\bar{L}$. If all guesses are correct, then output the string of b_x's over all $x \in \{0,1\}^n$ and accept; otherwise, halt and reject.

Note that the described nondeterministic algorithm will accept on at least one nondeterministic computation branch, and whenever it accepts, it outputs the truth table of L_n. Its running time is clearly $\mathsf{poly}(2^n)$.

Building a discrepancy set

By Theorem 93, given a 2^n-bit truth table of L_n, we get a collection of $t = 2^{n^c}$ binary strings of length $m = (s(n))^{1/c}$, for some constant $c > 0$, that is a discrepancy set for circuits of size m, for infinitely many input lengths n.

Running time analysis:

Using $n = m^{c/\alpha(n)}$, we get that $t \leq \exp(m^{d/\alpha(n)})$, for $d = c^2$. By Lemma 91, we may assume that $\alpha(n)$ satisfies the inequality $\alpha(n^k) \leq \alpha(n) + \lceil \log k \rceil$. Using this as well as the fact that $n \leq m$, we have for large enough m:

$$\alpha(n) = \alpha(m^{c/\alpha(n)}) \geq \alpha(m) - \lceil \log \alpha(n)/c \rceil \geq \alpha(m) - \lceil \log \alpha(m) \rceil \geq \alpha(m)/2.$$

We conclude that CAPP instances of size m are solvable i.o. in $\mathsf{NTIME}\left[2^{m^{2d/\alpha(m)}}\right]$.

Derandomizing promise-BPP **robustly often**

To boost this to the desired promise-$\mathsf{BPP} \subseteq \mathsf{ro}_\star$-$\mathsf{NSUBEXP}_c$ inclusion, we use the same "interval trick" as in the arithmetic case (Theorem 105, (2)). When asked to solve CAPP for a certain input length m, we go to the smallest length n such that $s(n) \geq m^c$, and consider truth tables for $L_{n'}$ over all lengths n' such that $n \leq n' \leq n^{\sqrt{\alpha(n)}}$. We concatenate all these truth tables together, getting a binary string of length at most $n^{\sqrt{\alpha(n)}} \cdot 2^{n^{\sqrt{\alpha(n)}}}$, which we view as the truth table of an n''-variate Boolean function, for $n'' = O(n^{\sqrt{\alpha(n)}})$.

We use the constructed truth table to construct a candidate discrepancy set via Theorem 93.

If the given interval above n contains a length n' such that $L_{n'}$ is hard, then our constructed truth table has hardness at least m^c, and so we get a true discrepancy set for circuits of size m, which allows us to solve CAPP for m-size instances. Since there infinitely many intervals containing a length n' where $L_{n'}$ is hard, there will be infinitely many intervals of superpolynomial length where our CAPP algorithm is correct.

The running time of the described algorithm will be dominated by the value t in Theorem 93 for $n'' = O(n^{\sqrt{\alpha(n)}})$. We get $t \leq 2^{(n'')^c} \leq \exp(n^{(c+1)\sqrt{\alpha(n)}}) \leq \exp(m^{c(c+1)/\sqrt{\alpha(n)}})$. Using our earlier lower bound $\alpha(n) \geq \alpha(m)/2$, we have $t \leq \exp(m^{c'/\sqrt{\alpha(m)}})$, for $c' = c(c+1)\sqrt{2}$, and so promise-BPP $\in$ ro$_\star$-NSUBEXP$_c$.

6.7.3 Robust Derandomization of CAPP Implies Robust Boolean Circuit Lower Bounds (forward direction of Theorem 110)

We argue by contradiction. Assume that CAPP is correctly approximated by a algorithm running in nondeterministic time $2^{n^{1/\beta(n)}}$, for some $\beta(n) \in \omega_c(1)$, on a ρ-robust set S for some $\rho(n) \in \omega_c(1)$. But, rp$_\star$-(NE$\cap$coNE) $\subseteq$ SIZE$[n^{\alpha(n)}]$ for every $\alpha(n) \in \omega_c(1)$.

First, we may assume, without loss of generality, that $\beta(n) = \rho(n)$, and moreover, for every $k \geq 1$, $\beta(n^k) \leq \beta(n) + \lceil \log k \rceil$. Indeed, if not, define $\delta'(n) := \min\{\beta(n), \rho(n)\}$, and use Lemma 91 to get $\delta(n) \leq \delta'(n)$ satisfying the property $\delta(n^k) \leq \delta(n) + \lceil \log k \rceil$ for all $k \geq 1$. It is easy to see that the δ-intervals around each $n \in \mathsf{core}(S)$ are contained within S, since $\delta(n) \leq \rho(n)$; so we get a δ-robust subset $S' \subseteq S$. Also, since $\delta(n) \leq \beta(n)$, we have $2^{n^{1/\beta(n)}} \leq 2^{n^{1/\delta(n)}}$. Hence, we get a nondeterministic $2^{n^{1/\delta(n)}}$-time algorithm correctly approximating CAPP on a δ-robust set S' of input lengths.

Next, similar to the proof of Theorem 109, implication (1), we follow three major steps: (i) collapsing E into ro$_\star$-NSUBEXP$_c$, (ii) using diagonalization to define a hard language in TIME$[2^{n^{\omega_c(1)}}]$, and (iii) using a padding argument to show that this hard language is actually computable in rp$_\star$-(NE$\cap$coNE), contradicting the assumption that rp$_\star$-(NE$\cap$coNE) has small circuits.

Collapsing E to $\mathsf{ro}_\star$-$\mathsf{NSUBEXP}_c$

First, $\mathsf{E} \subseteq \mathsf{rp}_\star\text{-}(\mathsf{NE} \cap \mathsf{coNE}) \subseteq \mathsf{SIZE}[n^{\alpha(n)}]$, for all $\alpha \in \omega_c(1)$. The last containment is by assumption, and the first containment follows by taking every input length as core and any function in $\omega_c(1)$ as the interval function. From this, it follows by [BFL91] that $\mathsf{E} \subseteq \mathsf{MA}[n^{c\alpha(n)}]$ for some constant $c > 0$. Now we use our CAPP algorithm to derandomize the MA class. The only difference here is that we must go through Lemma 96 to ensure that the derandomization works on a robust set.

For any $L \in \mathsf{MA}[n^{c\alpha(n)}]$, there is some $\mathsf{poly}(n^{c\alpha(n)}) = n^{cc'\alpha(n)}$ sized circuit with a witness and input wired in which takes only random bits as input and has the same probability of acceptance over those bits as the MA protocol run on input x with the given witness. We want to run CAPP on this circuit and accept if the probability of accepting is over the MA threshold. The size of input to CAPP on instance length n is therefore $t(n) = n^{cc'\alpha(n)}$, and so to place *MA* in $\mathsf{ro}_\star$-$\mathsf{NSUBEXP}_c$ we require that $S'' := t^{-1}(S')$ be computationally robust. Following the requirements of Lemma 96, we set $\gamma(n) := \sqrt{\delta(n)}/2$ and $\alpha(n) := \gamma(n)/(cc')$. Using our CAPP algorithm as described above, this places $\mathsf{MA}[n^{c\alpha(n)}] \subseteq \mathsf{NTIME}[2^{n^{1/\gamma(n)}}]$ on input lengths in γ-robust S''. Therefore, we have $\mathsf{E} \subseteq \mathsf{ro}_\star$-$\mathsf{NSUBEXP}_c$.

Diagonalization

This step is identical to the proof of Theorem 109. By standard diagonalization, we can get a language $L \in \mathsf{TIME}\left[2^{n^{\tau^2(n)}}\right] \notin \mathsf{io}\text{-}\mathsf{SIZE}[n^{\tau(n)}]$, for any $\tau \in \omega_c(1)$.

Padding

For the hard language L defined above, consider its graph version $M = \{(x,0) \mid x \notin L\} \cup \{(x,1) \mid x \in L\}$. Since L is computable in deterministic time $2^{n^{\tau^2(n)}}$, we conclude that M is computable within the same time bounds. Define the padded version $M_{pad} = \{(x, 1^{|x|^{\tau^2(|x|)} - |x|}) \mid x \in M\}$ of M. Since τ is efficiently computable, we conclude that $M_{pad} \in \mathsf{E} \subseteq \mathsf{NTIME}[2^{n^{1/\gamma(n)}}]$ on S''.

As in the proof of Theorem 106, to conclude $M \in \mathsf{rp}_\star\text{-}(\mathsf{NE} \cap \mathsf{coNE})$ from $M_{pad} \in \mathsf{ro}_\star\text{-}\mathsf{NSUBEXP}_c$ we need to pad by a superpolynomial amount, resulting in an input length of $t(n) = n^{\tau^2(n)}$ for the M_{pad} machine. To maintain the robustness of the set of inputs for which the M_{pad} algorithm works after this shift in input length, we use Lemma 97. Set $\gamma'(n) := \sqrt{\gamma(n)/2}$ and $\tau(n) := \sqrt{\gamma'(n)}$. Then by Lemma 97, $M \in \mathsf{NTIME}[2^{n^{1/\sqrt{\gamma'(n)}}}] \subseteq (\mathsf{NE} \cap \mathsf{coNE})$ on some γ'-robust set. This is exactly $M \in \mathsf{rp}_\star\text{-}(\mathsf{NE} \cap \mathsf{coNE})$, and gives us the contradiction: we assumed $\mathsf{rp}_\star\text{-}(\mathsf{NE} \cap \mathsf{coNE}) \subseteq \mathsf{SIZE}[n^{\alpha(n)}]$ for every $\alpha \in \omega_c(1)$, but $M \notin \mathsf{SIZE}[n^\tau]$ and $\tau \in \omega_c(1)$, concluding the proof.

6.7.4 Robust Boolean Circuit Lower Bounds Imply Robust Derandomization of CAPP (backwards direction of Theorem 110)

Assume that there is some promise problem Π in $\mathsf{rp}_\star\text{-}(\mathsf{NE} \cap \mathsf{coNE})$ on computably robust set S such that, for some computably superpolynomial function $s(n) = n^{\alpha(n)}$, there are infinitely many input lengths $n \in \mathsf{core}(S)$ where Π_n cannot be computed by a Boolean circuit of size $s(n)$.

As in the proof of Theorem 109, we will use Theorem 93 to get a discrepency set to derandomize CAPP. Similar to the proof of Theorem 109, we search an interval for hard slices to obtain derandomization robustly often. The only difference is that we are not free to set the size of the interval that we search.

Building a discrepency set by searching intervals

Let G be the "good" set of input lengths obtained by applying Lemma 97 to S. Then, we have that G is exactly the set of input lengths n where an interval around a $n^{o_c(1)}$ function of n is guaranteed to contain a hard length for Π. Suppose we are given a CAPP instance of size m, for $m \in G$. We use the following algorithm:

1. Non-deterministically construct the truth table of $\Pi_{m'}$, for each $m' \in I[m]$.
2. Concatenate together all the above truth tables into one string. By definition of $I[m]$, the length of this string is upper-bounded by $T_h(m) \cdot 2^{T_h(m)}$.

3. Treating the resulting string as the truth table of a $O(T_h(m))$-variate Boolean function, use Theorem 93 to get a discrepency set and derandomize the CAPP instance

For step 1, use the method given in the proof of Theorem 109 for nondeterminstically generating truth tables. The step will succeed on some path if each $m' \in I[m]$ is also in S, which we have by definition of G from Lemma 97.

We now analyse the application of Theorem 93 to the concatenated truth tables. There are $t = 2^{O(T_h(m))}$ binary strings in the resulting set. Say that the hard slice for Π is in the worst-case location for the hardness measure, the smallest element of $I[m]$ which is $T_l(m)$. Then $s(T_l(m))^{1/c}$ is a lower bound on the size of circuits that the discrepency set will fool. But Lemma 97 tells us that $s(T_l(m)) = T_l(m)^{\alpha(T_l(m)} \in m^{\omega_c(1)}$, and $(m^{\omega_c(1)})^{1/c}$ remains in $m^{\omega_c(1)} > m$. Therefore, the resulting set of strings is always discrepency set for circuits of size $m \in G$.

As in Theorem 109, the runtime of our derandomization is dominated by $t = 2^{O(T_h(m))^c}$, the number of strings in the discrepency set. By Lemma 97, we have that $T_h(m) \in m^{o_c(1)}$. By elementary closure properties of the computably super-constant functions, $O(m^{o_c(1)})^c \in m^{o_c(1)}$. Therefore, there is a NSUBEXP time algorithm for CAPP on the computably robust set G. This concludes proof of the theorem.

6.8 Robustly-often nontrivial useful properties

A *property* of Boolean functions is a family $\mathscr{P} = \{\mathscr{P}_n\}_{n \geq 0}$ of predicates $\mathscr{P}_n : \{0,1\}^{2^n} \to \{0,1\}$. We think of $\mathscr{P}_n$ as taking the truth table of an n-variate Boolean function as input. We say that $\mathscr{P}$ is NP-*computable* if there exists a nondeterministic algorithm that computes each $\mathscr{P}_n$ in time $\mathsf{poly}(2^n)$, for all sufficiently large $n \in \mathbb{N}$.

For a function $s : \mathbb{N} \to \mathbb{N}$, we say that a property $\mathscr{P}$ is *useful against* $\mathsf{SIZE}[s]$ *at length* n if, whenever $\mathscr{P}_n$ accepts the truth table of a Boolean n-variate function $f_n : \{0,1\}^n \to \{0,1\}$, this means that f_n requires circuit size at least $s(n)$. We say that $\mathscr{P}$ is *nontrivial at length* n if $\mathscr{P}_n$ accepts at least one truth table of length 2^n.

Finally, we say that $\mathscr{P}$ is *robustly-often nontrivially useful against* $\mathsf{SIZE}[\mathsf{superpoly}_c]$ *(denoted* $\mathsf{ro}_\star$*-useful)* if, for some $s(n) \in \mathsf{superpoly}_c$,

1. $S = \{n \in \mathbb{N} \mid \ \mathscr{P}$ is nontrivial at length $n\}$ is computably robust, and
2. $\mathscr{P}$ is useful against $\mathsf{SIZE}[s]$ at length n for infinitely many lengths $n \in \mathsf{core}(S)$.

As a corollary of Theorem 110, we get the following equivalence between circuit lower bounds for $\mathsf{NEXP} \cap \mathsf{coNEXP}$ and the existence of $\mathsf{ro}_\star$-useful properties.

Theorem 111 (Equivalence Between Circuit Lower Bounds and Useful Properties)**.**

$$\mathsf{rp}_\star\text{-}(\mathsf{NE} \cap \mathsf{coNE}) \not\subseteq \mathsf{SIZE}[\mathsf{superpoly}_c] \Leftrightarrow \exists\ \mathsf{NP}\text{-}\textit{computable}\ \mathsf{ro}_\star\text{-}\textit{useful property}\ \mathscr{P}$$

Proof. ($\Rightarrow$) Assume we have a pair of NE machines M_1 and M_0 that correctly compute L and $\bar{L}$, respectively, for some computably robust set S of input lengths, where L requires $\mathsf{superpoly}_c$ circuit size for infinitely many input lengths in $\mathsf{core}(S)$. Define a property $\mathscr{P}$ as follows:

> "On input $T \in \{0,1\}^{2^n}$, guess 2^n candidate witnesses $a_1, \ldots, a_{2^n}$ of length $2^{O(n)}$ each, and check, for every $1 \leq i \leq 2^n$, if $T_i = b$, for $b \in \{0,1\}$, then $M_b(i)$ accepts a_i as a witness. If succeed for all i's, then accept. Otherwise, reject."

Clearly, $\mathscr{P}$ is NP-computable. Note that $\mathscr{P}_n$ accepts the truth table of L_n (and nothing else) for the computably robust set S of input lengths n, and so $\mathscr{P}$ is robustly-often nontrivial. Finally, since L requires circuit size $\mathsf{superpoly}_c$ foro infinitely many input lengths $n \in \mathsf{core}(S)$, we get that $\mathscr{P}$ is useful against $\mathsf{SIZE}[\mathsf{superpoly}_c]$ at length n for infinitely many $n \in \mathsf{core}(S)$. Thus $\mathscr{P}$ is $\mathsf{ro}_\star$-useful.

($\Leftarrow$) Given an NP-computable property $\mathscr{P}$ that is nontrivial on a computably robust set S of input lengths n, and useful against $\mathsf{SIZE}[\mathsf{superpoly}_c]$ for infinitely many $n \in \mathsf{core}(S)$, we use $\mathscr{P}$ to nondeterministically guess a truth table T of length 2^n, nondeterministically verify that T is accepted by $\mathscr{P}_n$, and use T as a "hard" Boolean function to derandomize promise-BPP (using the hardness-randomness tradeoff of Theorem 93).

We get that promise-$\mathsf{BPP} \subseteq$ io-$\mathsf{NSUBEXP}_c$, since infinitely often we get a truth table T of superpolynomial circuit complexity. Using the "interval trick" (as in the proof of Theorem 109, implication (2)), we boost this inclusion to get promise-$\mathsf{BPP} \subseteq \mathsf{ro}_\star$-$\mathsf{NSUBEXP}_c$, which, by Theorem 110, implies $\mathsf{rp}_\star$-$(\mathsf{NE} \cap \mathsf{coNE}) \not\subseteq \mathsf{SIZE}[\mathsf{superpoly}_c]$. □

We also get a more general theorem that involves *fully deterministic* properties in the equivalence.

Theorem 112. *The following are equivalent:*

1. $\mathsf{rp}_\star$-$(\mathsf{NE} \cap \mathsf{coNE}) \not\subseteq \mathsf{SIZE}[\mathsf{superpoly}_c]$
2. *there is an* NP*-computable* $\mathsf{ro}_\star$*-useful property*
3. *there is a* P*-computable* $\mathsf{ro}_\star$*-useful property*

Proof. We prove that $(1) \Rightarrow (2) \Rightarrow (3) \Rightarrow (1)$.

- **(1) ⇒ (2):** Assume we have a pair of NE machines M_1 and M_0 that correctly compute L and $\bar{L}$, respectively, for some computably robust set S of input lengths, where L requires $\mathsf{superpoly}_c$ circuit size for infinitely many input lengths in $\mathsf{core}(S)$. Define a property $\mathscr{P}$ as follows:

  > "On input $T \in \{0,1\}^{2^n}$, guess 2^n candidate witnesses $a_1, \ldots, a_{2^n}$ of length $2^{O(n)}$ each, and check, for every $1 \leq i \leq 2^n$, if $T_i = b$, for $b \in \{0,1\}$, then $M_b(i)$ accepts a_i as a witness. If succeed for all i's, then accept. Otherwise, reject."

  Clearly, $\mathscr{P}$ is NP-computable. Note that $\mathscr{P}_n$ accepts the truth table of L_n (and nothing else) for the computably robust set S of input lengths n, and so $\mathscr{P}$ is robustly-often nontrivial. Finally, since L requires circuit size $\mathsf{superpoly}_c$ for infinitely many input lengths $n \in \mathsf{core}(S)$, we get that $\mathscr{P}$ is useful against $\mathsf{SIZE}[\mathsf{superpoly}_c]$ at length n for infinitely many $n \in \mathsf{core}(S)$. Thus $\mathscr{P}$ is $\mathsf{ro}_\star$-useful.

- **(2) $\Rightarrow$ (3):** Assume we have a NP-computable $\mathsf{ro}_\star$-useful property $\mathscr{P}$. Let $R(x,y)$ be the witness-checking polynomial-time predicate of the NP algorithm defining $\mathscr{P}$: $\forall x \in \{0,1\}^{2^n}\ x \in \mathscr{P} \Leftrightarrow \exists y \in \{0,1\}^{2^{cn}}\ R(x,y)$. Define a new property $\mathscr{P}'$ to accept a string z of length $2^n + 2^{cn}$ iff $z = xy$ such that $R(x,y)$ is true. This is clearly P-computable. Next we argue that $\mathscr{P}'$ is $\mathsf{ro}_\star$-useful.

  Let S be the computably robust set where $\mathscr{P}$ is nontrivial. Let $\alpha \in \omega_c(1)$ be such that there are infinitely many $m \in \mathsf{core}(S)$ so that, every m' in the interval $m^{1/\alpha(m)} \leq m' \leq m^{\alpha(m)}$ is in S. Set $\beta(m) = \alpha(m)/2$; we have $\beta(m) \in \omega_c(1)$ since computably super-constant functions are closed under multiplication by constants. Define the robust set S' by taking β-intervals around cm for all $m \in \mathsf{core}(S)$. We get that $\mathsf{core}(S') = \{cm \mid m \in \mathsf{core}(S)\}$.

  We claim that $\mathscr{P}'$ is a $\mathsf{ro}_\star$-useful property on S'.

  First, observe that $\mathscr{P}'$ at input length n depends on $\mathscr{P}$ at length $\lfloor n/c \rfloor$. Since $\mathscr{P}$ is useful at length m for infinitely many $m \in \mathsf{core}(S)$, we get that $\mathscr{P}'$ is also useful (for a slightly smaller, but still superpolynomial circuit size bound) for infinitely many $cm \in \mathsf{core}(S')$.

  Next we show that $\mathscr{P}'$ is nontrivial at all lengths $n \in S'$. Take an arbitrary n inside the β-interval around cm for some $m \in \mathsf{core}(S)$. We have

$$(cm)^{1/\beta(m)} \leq n \leq (cm)^{\beta(m)}. \tag{6.2}$$

  We get that $\mathscr{P}'$ is nontrivial at length n if $m^{1/\alpha(m)} \leq n/c \leq m^{\alpha(m)}$, which is equivalent to $cm^{1/\alpha(m)} \leq n \leq cm^{\alpha(m)}$. By Eq. (6.2), we have $n \leq (cm)^{\alpha(m)/2} \leq cm^{\alpha(m)}$ for sufficiently large m. We also have $cm^{1/\alpha(m)} \leq (cm)^{2/\alpha(m)}$ when $m^{1/\alpha(m)} \geq c$, or equivalently, when $\alpha(m) \leq (\log m)/(\log c)$; we can assume the latter by using $\alpha'(m) := \min\{\alpha(m), (\log m)/(\log c)\}$ instead of $\alpha(m)$ if necessary. We conclude that $\mathscr{P}'$ is nontrivial at every n satisfying Eq. (6.2).

- **(3) $\Rightarrow$ (1):** Given an P-computable property $\mathscr{P}$ that is nontrivial on a computably robust set

S of input lengths n, and useful against $\mathsf{SIZE}[\mathsf{superpoly}_c]$ for infinitely many $n \in \mathsf{core}(S)$, we use $\mathscr{P}$ to nondeterministically guess a truth table T of length 2^n, verify that T is accepted by $\mathscr{P}_n$, and use T as a "hard" Boolean function to derandomize promise-BPP (using the hardness-randomness tradeoff of Theorem 93).

We get that promise-BPP $\subseteq$ io-NSUBEXP$_c$, since infinitely often we get a truth table T of superpolynomial circuit complexity. Using the "interval trick" (as in the proof of Theorem 109, implication (2)), we boost this inclusion to get promise-BPP $\subseteq$ ro$_\star$-NSUBEXP$_c$, which, by Theorem 110, implies rp$_\star$-(NE $\cap$ coNE) $\not\subseteq$ SIZE[superpoly$_c$].

□

Chapter 6, in part, is based on the material as it appears in "Marco Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Tighter connections between derandomization and circuit lower bounds. In Naveen Garg, Klaus Jansen, Anup Rao, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA*, volume 40 of *LIPIcs*, pages 645–658. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015". The dissertation author was the primary investigator and author of this paper.

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[ABW15] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is valiant's parser. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 98–117. IEEE Computer Society, 2015.

[AHWW15] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends or: A polylog shaved is a lower bound made. *CoRR*, abs/1511.06022, 2015.

[Ale03] Michael Alekhnovich. More on average case vs approximation complexity. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 298–307. IEEE, 2003.

[AM11] S. Aaronson and D. van Melkebeek. On circuit lower bounds from derandomization. *Theory of Computing*, 7(1):177–184, 2011.

[AvM12] B. Aydinlioglu and D. van Melkebeek. Nondeterministic circuit lower bounds from mildly de-randomizing arthur-merlin games. In *Proceedings of the 27th Conference on Computational Complexity, CCC 2012, Porto, Portugal, June 26-29, 2012*, pages 269–279, 2012.

[AW85] M. Ajtai and A. Wigderson. Deterministic simulation of probabilistic constant depth circuits. In *Proceedings of the Twenty-Sixth Annual IEEE Symposium on Foundations of Computer Science*, pages 11–19, 1985.

[AWW14] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2014.

[BDT16] Arturs Backurs, Nishanth Dikkala, and Christos Tzamos. Tight hardness results for maximum weight rectangles. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 81:1–81:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[BFKL93] Avrim Blum, Merrick Furst, Michael Kearns, and Richard J Lipton. Cryptographic primitives based on hard learning problems. In *Annual International Cryptology Conference*, pages 278–291. Springer, 1993.

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

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

[BGL17] Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 307–318. IEEE Computer Society, 2017.

[BI15] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 51–58. ACM, 2015.

[BIS90] David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within nc^1. *J. Comput. Syst. Sci.*, 41(3):274–306, 1990.

[BIS12] Paul Beame, Russell Impagliazzo, and Srikanth Srinivasan. Approximating AC^0 by small height decision trees and a deterministic algorithm for $\#AC^0$SAT. In *Proceedings of the 27th Conference on Computational Complexity, CCC 2012, Porto, Portugal, June 26-29, 2012*, pages 117–125, 2012.

[BK15] Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97. IEEE Computer Society, 2015.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM (JACM)*, 50(4):506–519, 2003.

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864, 1984.

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

[Bop97] Ravi B. Boppana. The average sensitivity of bounded-depth circuits. *Information Processing Letters*, 63(5):257–261, September 1997.

[Bra10] Mark Braverman. Polylogarithmic independence fools AC^0 circuits. *Journal of the Association for Computing Machinery*, 57:28:1–28:10, 2010.

[BRSV17] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Average-case fine-grained hardness. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 483–496. ACM, 2017.

[BT06a] Andrej Bogdanov and Luca Trevisan. Average-case complexity. *Foundations and Trends in Theoretical Computer Science*, 2(1), 2006.

[BT06b] Andrej Bogdanov and Luca Trevisan. On worst-case to average-case reductions for NP problems. *SIAM J. Comput.*, 36(4):1119–1159, 2006.

[BT17] Arturs Backurs and Christos Tzamos. Improving viterbi is hard: Better runtimes imply faster clique algorithms. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 311–321. PMLR, 2017.

[CIKK15] Marco Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Tighter connections between derandomization and circuit lower bounds. In Naveen Garg, Klaus Jansen, Anup Rao, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA*, volume 40 of *LIPIcs*, pages 645–658. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[CIKK16] Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Learning algorithms from natural proofs. In Ran Raz, editor, *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, volume 50 of *LIPIcs*, pages 10:1–10:24. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[CIKK17] Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Agnostic learning from tolerant natural proofs. In Klaus Jansen,

José D. P. Rolim, David Williamson, and Santosh Srinivas Vempala, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, volume 81 of *LIPIcs*, pages 35:1–35:19. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[CIS18] Marco L. Carmosino, Russell Impagliazzo, and Manuel Sabin. Fine-grained derandomization: From problem-centric to resource-centric complexity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 27:1–27:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[CK15] Ruiwen Chen and Valentine Kabanets. Correlation bounds and #sat algorithms for small linear-size circuits. In Dachuan Xu, Donglei Du, and Dingzhu Du, editors, *Computing and Combinatorics - 21st International Conference, COCOON 2015, Beijing, China, August 4-6, 2015, Proceedings*, volume 9198 of *Lecture Notes in Computer Science*, pages 211–222. Springer, 2015.

[CKK+15] Ruiwen Chen, Valentine Kabanets, Antonina Kolokolova, Ronen Shaltiel, and David Zuckerman. Mining circuit lower bound proofs for meta-algorithms. *Computational Complexity*, 24(2):333–392, 2015.

[CKS14] Ruiwen Chen, Valentine Kabanets, and Nitin Saurabh. An improved deterministic #SAT algorithm for small de Morgan formulas. In *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, pages 165–176, 2014.

[Cob65] Alan Cobham. The intrinsic computational difficulty of functions. In Yehoshua Bar-Hillel, editor, *Logic, Methodology and Philosophy of Science: Proceedings of the 1964 International Congress (Studies in Logic and the Foundations of Mathematics)*, pages 24–30. North-Holland Publishing, 1965.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[CS15] Ruiwen Chen and Rahul Santhanam. Improved algorithms for sparse MAX-SAT and MAX-k-CSP. In *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, pages 33–45, 2015.

[DL78] R.A. DeMillo and R.J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7:193–195, 1978.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[EG04] Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theor. Comput. Sci.*, 326(1-3):57–67, 2004.

[Fel09] Vitaly Feldman. On the power of membership queries in agnostic learning. *Journal of Machine Learning Research*, 10:163–182, 2009.

[Fel10] Vitaly Feldman. Distribution-specific agnostic boosting. In *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*, pages 241–250, 2010.

[FF91] Joan Feigenbaum and Lance Fortnow. On the random-self-reducibility of complete sets. In *Proceedings of the Sixth Annual Structure in Complexity Theory Conference, Chicago, Illinois, USA, June 30 - July 3, 1991*, pages 124–132. IEEE Computer Society, 1991.

[FF93] Joan Feigenbaum and Lance Fortnow. Random-self-reducibility of complete sets. *SIAM J. Comput.*, 22(5):994–1005, 1993.

[FGKP06] Vitaly Feldman, Parikshit Gopalan, Subhash Khot, and Ashok Kumar Ponnuswami. New results for learning noisy parities and halfspaces. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 563–574. IEEE, 2006.

[FK09] Lance Fortnow and Adam R. Klivans. Efficient learning algorithms yield circuit lower bounds. *J. Comput. Syst. Sci.*, 75(1):27–36, 2009.

[FS96] Jean-Bernard Fischer and Jacques Stern. An efficient pseudo-random generator provably as secure as syndrome decoding. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 245–255. Springer, 1996.

[FS11] L. Fortnow and R. Santhanam. Robust simulations and significant separations. In *Automata, Languages and Programming - 38th International Colloquium, ICALP, Proceedings, Part I*, pages 569–580, 2011.

[FSS84] Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

[FSUV13] Bill Fefferman, Ronen Shaltiel, Christopher Umans, and Emanuele Viola. On beating the hybrid argument. *Theory of Computing*, 9:809–843, 2013.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[GI16] Jiawei Gao and Russell Impagliazzo. Orthogonal vectors is hard for first-order properties on sparse graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:53, 2016.

[GIKW17] Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and R. Ryan Williams. Completeness for first-order properties on sparse structures with algorithmic applications. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2162–2181. SIAM, 2017.

[Gil77] J. Gill. Computational complexity of probabilistic turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.

[GKK08] Parikshit Gopalan, Adam Tauman Kalai, and Adam R. Klivans. Agnostically learning decision trees. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 527–536. ACM, 2008.

[GKL93] Oded Goldreich, Hugo Krawczyk, and Michael Luby. On the existence of pseudorandom generators. *SIAM J. Comput.*, 22(6):1163–1175, 1993.

[GKS95] Sally A. Goldman, Michael J. Kearns, and Robert E. Schapire. On the sample complexity of weakly learning. *Inf. Comput.*, 117(2):276–287, 1995.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 25–32. ACM, 1989.

[GNW11] Oded Goldreich, Noam Nisan, and Avi Wigderson. On Yao's XOR-lemma. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 273–301. Springer, 2011.

[GR18] Oded Goldreich and Guy N. Rothblum. Counting t-cliques: Worst-case to average-case reductions and direct interactive proof systems. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:46, 2018.

[GRS00] Oded Goldreich, Ronitt Rubinfeld, and Madhu Sudan. Learning polynomials with queries: The highly noisy case. *SIAM J. Discrete Math.*, 13(4):535–570, 2000.

[GST03] Dan Gutfreund, Ronen Shaltiel, and Amnon Ta-Shma. Uniform hardness vs. randomness tradeoffs for arthur-merlin games. In *18th Annual IEEE Conference on Computational Complexity (Complexity 2003), 7-10 July 2003, Aarhus, Denmark*, pages 33–47. IEEE Computer Society, 2003.

[GST07] Dan Gutfreund, Ronen Shaltiel, and Amnon Ta-Shma. If NP languages are hard on the worst-case, then it is easy to find their hard instances. *Computational Complexity*, 16(4):412–441, 2007.

[GW02] Oded Goldreich and Avi Wigderson. Derandomization that is rarely wrong from short advice that is typically good. In José D. P. Rolim and Salil P. Vadhan, editors,

Randomization and Approximation Techniques, 6th International Workshop, RANDOM 2002, Cambridge, MA, USA, September 13-15, 2002, Proceedings, volume 2483 of *Lecture Notes in Computer Science*, pages 209–223. Springer, 2002.

[Hås87] Johan Håstad. *Computational Limitations of Small-depth Circuits*. MIT Press, Cambridge, MA, USA, 1987.

[Hås89] Johan Håstad. Almost optimal lower bounds for small depth circuits. In S. Micali, editor, *Randomness and Computation*, pages 143–170, Greenwich, Connecticut, 1989. Advances in Computing Research, vol. 5, JAI Press.

[Hås01] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM (JACM)*, 48(4):798–859, 2001.

[HB01] Nicholas J. Hopper and Manuel Blum. Secure human identification protocols. In *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, pages 52–66, 2001.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364 – 1396, 1999.

[Hir18] Shuichi Hirahara. Non-black-box worst-case to average-case reductions within NP. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:138, 2018.

[HS65] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[HS82] J. Heintz and C.-P. Schnorr. Testing polynomials which are easy to compute. *L'Enseignement Mathématique*, 30:237–254, 1982.

[HVV06] Alexander Healy, Salil Vadhan, and Emanuele Viola. Using nondeterminism to amplify hardness. *SIAM Journal on Computing*, 35(4):903–931, 2006.

[IJKW10] Russell Impagliazzo, Ragesh Jaiswal, Valentine Kabanets, and Avi Wigderson. Uniform direct product theorems: Simplified, optimized, and derandomized. *SIAM J. Comput.*, 39(4):1637–1665, 2010.

[IKV18] Russell Impagliazzo, Valentine Kabanets, and Ilya Volkovich. The power of natural properties as oracles. In Rocco A. Servedio, editor, *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, volume 102 of *LIPIcs*, pages 7:1–7:20. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

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

[IMP12] Russell Impagliazzo, William Matthews, and Ramamohan Paturi. A satisfiability algorithm for ac^0. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 961–972. SIAM, 2012.

[IMZ12] Russell Impagliazzo, Raghu Meka, and David Zuckerman. Pseudorandomness from shrinkage. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 111–119. IEEE Computer Society, 2012.

[IW97] Russell Impagliazzo and Avi Wigderson. *P = BPP* if *E* requires exponential circuits: Derandomizing the XOR lemma. In Frank Thomson Leighton and Peter W. Shor, editors, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 220–229. ACM, 1997.

[IW01] Russell Impagliazzo and Avi Wigderson. Randomness vs time: Derandomization under a uniform assumption. *J. Comput. Syst. Sci.*, 63(4):672–688, 2001.

[Jac06] Jeffrey C. Jackson. Uniform-distribution learnability of noisy linear threshold functions with restricted focus of attention. In *Proceedings of the 19th Annual Conference on Learning Theory*, COLT'06, pages 304–318, Berlin, Heidelberg, 2006. Springer-Verlag.

[JMV15] Hamid Jahanjou, Eric Miles, and Emanuele Viola. Local reductions. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 749–760. Springer, 2015.

[JS12] M.J. Jansen and R. Santhanam. Stronger lower bounds and randomness-hardness trade-offs using associated algebraic complexity classes. In Christoph Dürr and Thomas Wilke, editors, *STACS*, volume 14 of *LIPIcs*, pages 519–530. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

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

[Kal89] E. Kaltofen. Factorization of polynomials given by straight-line programs. In S. Micali, editor, *Randomness and Computation*, volume 5 of *Advances in Computing Research*, pages 375–412. JAI Press, Greenwich, CT, 1989.

[Kan82] R. Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, 55(1-3):40–56, 1982.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas*

J. Watson Research Center, Yorktown Heights, New York., The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

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

[KKMS08] Adam Tauman Kalai, Adam R. Klivans, Yishay Mansour, and Rocco A. Servedio. Agnostically learning halfspaces. *SIAM J. Comput.*, 37(6):1777–1805, 2008.

[KKO13] Adam Klivans, Pravesh Kothari, and Igor Carboni Oliveira. Constructing hard functions using learning algorithms. In *Proceedings of the 28th Conference on Computational Complexity, CCC 2013, Palo Alto, California, USA, 5-7 June, 2013*, pages 86–97, 2013.

[KL82] R.M. Karp and R.J. Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28(3-4):191–209, 1982.

[KMS12] J. Kinne, D. van Melkebeek, and R. Shaltiel. Pseudorandom generators, typically-correct derandomization, and circuit lower bounds. *Computational Complexity*, 21(1):3–61, 2012.

[KSS94] Michael J. Kearns, Robert E. Schapire, and Linda Sellie. Toward efficient agnostic learning. *Machine Learning*, 17(2-3):115–141, 1994.

[KvMS12] Jeff Kinne, Dieter van Melkebeek, and Ronen Shaltiel. Pseudorandom generators, typically-correct derandomization, and circuit lower bounds. *Computational Complexity*, 21(1):3–61, 2012.

[KW17] Daniel M. Kane and R. Ryan Williams. The orthogonal vectors conjecture for branching programs and formulas. *CoRR*, abs/1709.05294, 2017.

[LBW96] Wee Sun Lee, Peter L. Bartlett, and Robert C. Williamson. Efficient agnostic learning of neural networks with bounded fan-in. *IEEE Trans. Information Theory*, 42(6):2118–2132, 1996.

[LF06] Éric Levieil and Pierre-Alain Fouque. An improved lpn algorithm. In *International Conference on Security and Cryptography for Networks*, pages 348–359. Springer, 2006.

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

[Lin18] Andrea Lincoln. Personal communication, 2018.

[Lip89] Richard J. Lipton. New directions in testing. In Joan Feigenbaum and Michael Merritt, editors, *Distributed Computing And Cryptography, Proceedings of a DIMACS Workshop, Princeton, New Jersey, USA, October 4-6, 1989*, volume 2 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 191–202. DIMACS/AMS, 1989.

[LMN93] Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, fourier transform, and learnability. *J. ACM*, 40(3):607–620, 1993.

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

[Lup58] Oleg B. Lupanov. On the synthesis of switching circuits. *Soviet Mathematics*, 119(1):23–26, 1958. English translation in *Soviet Mathematics Doklady*.

[Lup59] Oleg B. Lupanov. A method of circuit synthesis. *Izvestiya VUZ, Radiofizika*, 1(1):120–140, 1959. (in Russian).

[Lyu05] Vadim Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 378–389. Springer, 2005.

[NP85] Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994.

[O'D04] Ryan O'Donnell. Hardness amplification within np. *J. Comput. Syst. Sci.*, 69(1):68–94, 2004.

[OS17] Igor Carboni Oliveira and Rahul Santhanam. Conspiracies between learning algorithms, circuit lower bounds, and pseudorandomness. In Ryan O'Donnell, editor, *32nd Computational Complexity Conference, CCC 2017, July 6-9, 2017, Riga, Latvia*, volume 79 of *LIPIcs*, pages 18:1–18:49. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[OS18] Igor Carboni Oliveira and Rahul Santhanam. Pseudo-derandomizing learning and approximation. In Eric Blais, Klaus Jansen, José D. P. Rolim, and David Steurer, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018, August 20-22, 2018 - Princeton, NJ, USA*, volume 116 of *LIPIcs*, pages 55:1–55:19. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[Pie12] Krzysztof Pietrzak. Cryptography from learning parity with noise. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 99–114. Springer, 2012.

[PW86] M. Paterson and I. Wegener. Nearly optimal hierarchies for network and formula size. *Acta Informatica*, 23:217–221, 1986.

[Raz87] Alexander A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes*, 41(4):333–338, 1987.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

[RR97] Alexander A. Razborov and Steven Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, 1997.

[San10] Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 183–192. IEEE Computer Society, 2010.

[Sch80] J.T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the Association for Computing Machinery*, 27(4):701–717, 1980.

[Sha38] C. E. Shannon. A symbolic analysis of relay and switching circuits. *Transactions of the American Institute of Electrical Engineers*, 57(12):713–723, Dec 1938.

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

[Sha11] Ronen Shaltiel. Weak derandomization of weak algorithms: Explicit versions of yao's lemma. *Computational Complexity*, 20(1):87–143, 2011.

[She92] Alexander Shen. IP = PSPACE: simplified proof. *J. ACM*, 39(4):878–880, 1992.

[Smo87] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 77–82, 1987.

[Sri15] Srikanth Srinivasan. A compression algorithm for $AC^0[\oplus]$ circuits using certifying polynomials. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:142, 2015.

[ST12] Kazuhisa Seto and Suguru Tamaki. A satisfiability algorithm and average-case hardness for formulas over the full binary basis. In *Proceedings of the Twenty-Seventh Annual IEEE Conference on Computational Complexity*, pages 107–116, 2012.

[STV01] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.

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

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

[SW15] Rahul Santhanam and Richard Ryan Williams. Beating exhaustive search for quantified boolean formulas and connections to circuit complexity. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 231–241, 2015.

[Tal15] Avishay Tal. #SAT algorithms from shrinkage. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:114, 2015.

[Tod91] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[Tra84] Boris A. Trakhtenbrot. A survey of russian approaches to perebor (brute-force searches) algorithms. *IEEE Annals of the History of Computing*, 6(4):384–400, 1984.

[Tre05] Luca Trevisan. On uniform amplification of hardness in NP. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 31–38. ACM, 2005.

[Tur36] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936.

[TV02] Luca Trevisan and Salil P. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. In *Proceedings of the 17th Annual IEEE Conference on Computational Complexity, Montréal, Québec, Canada, May 21-24, 2002*, pages 129–138. IEEE Computer Society, 2002.

[TV07] Luca Trevisan and Salil P. 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. *J. Comput. Syst. Sci.*, 67(2):419–440, 2003.

[Val79] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[Val84] Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, November 1984.

[vN51] John von Neumann. Various techniques used in connection with random digits. *J. Research Nat. Bur. Stand., Appl. Math. Series*, 12:36–38, 1951.

[Wil05] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.

[Wil10] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 231–240. ACM, 2010.

[Wil11] Ryan Williams. Non-uniform ACC circuit lower bounds. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity, CCC 2011, San Jose, California, June 8-10, 2011*, pages 115–125. IEEE Computer Society, 2011.

[Wil13] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM J. Comput.*, 42(3):1218–1244, 2013.

[Wil14a] Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 194–202, 2014.

[Wil14b] Ryan Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2:1–2:32, 2014.

[Wil15] Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In Thore Husfeldt and Iyad A. Kanj, editors, *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, volume 43 of *LIPIcs*, pages 17–29. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[Wil16] Richard Ryan Williams. Strong ETH breaks with merlin and arthur: Short non-interactive proofs of batch evaluation. In Ran Raz, editor, *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, volume 50 of *LIPIcs*, pages 2:1–2:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 80–91. IEEE Computer Society, 1982.

[Zip79] R.E. Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of an International Symposium on Symbolic and Algebraic Manipulation (EUROSAM'79)*, Lecture Notes in Computer Science, pages 216–226, 1979.